AFRL-RI-RS-TR-2015-139

CONFABULATION BASED REAL-TIME ANOMALY DETECTION FOR WIDE-AREA SURVEILLANCE USING HETEROGENEOUS HIGH PERFORMANCE COMPUTING ARCHITECTURE

SYRACUSE UNIVERSITY

JUNE 2015

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2015-139 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/ S /
GEORGE O. RAMSEYER
Work Unit Manager

/ S /
MARK H. LINDERMAN
Technical Advisor, Computing
& Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
JUNE 2015	FINAL TECHNICAL REPORT	JUN 2012 – DEC 2014

4. TITLE AND SUBTITLE

CONFABULATION BASED REAL-TIME ANOMALY DETECTION FOR WIDE-AREA SURVEILLANCE USING HETEROGENEOUS HIGH PERFORMANCE COMPUTING ARCHITECTURE

5a. CONTRACT NUMBER
FA8750-12-1-0251

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62788F

6. AUTHOR(S)

Qinru Qiu

5d. PROJECT NUMBER
925F

5e. TASK NUMBER
OS

5f. WORK UNIT NUMBER
YR

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Syracuse University
CST 4-206
Syracuse, NY 13244

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RITB
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2015-139

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The feasibility of probabilistic inference based anomaly detection was determined, and those results were applied to wide area surveillance. An abstract-level autonomous information processing framework was developed that provided continuous monitoring and real-time anomaly detection over hundreds of square kilometer areas. The anomaly recognition and detection (AnRAD) system was built as a cogent confabulation network. It represented road traffic using a set of features extracted from a Ground Moving Target Indicator (GMTI) input stream and performed likelihood-ratio testing on a set of key features to detect abnormal vehicle behavior. Due to its low learning and recall complexities, the AnRAD supported incremental learning, which was proved to enhance the detection accuracy. A self-structuring technique was developed that learned the structure of a probabilistic inference network from unlabeled data. Without any assumption of the distribution of data, mutual information between features was leveraged to learn a succinct network configuration. Compared to several existing anomaly detection methods, the proposed approach provided higher detection performances and excellent reasoning capabilities. Massive parallelism was inherent to the inference model and accelerated the detection process using state-of-the-art multicore processors including graphic processor units (GPUs) and Intel Xeon Phi processors. Experimental results showed significant speedups, which can enable real-time applications with high-volume data streams.

15. SUBJECT TERMS
Neuromorphic, anomaly detection , confabulation, high performance computing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**GEORGE RAMSEYER**
U	U	U	SAR	47	19b. TELEPHONE NUMBER (*Include area code*) **N/A**

Table of Contents

List of Figures iii
List of Tables iv
1 Summary 1
2 Introduction 2
2.1 Overview 2
2.2 Confabulation-based Network Architecture 3
2.3 Cogent Confabulation 3
3 Methods, Assumptions, and Procedures 5
3.1 Zone Partition 5
3.2 Confabulation Networks 7
3.2.1 Confabulation Network Self-structuring. 8
3.2.2 Key Node Hierarchy. 8
3.2.3 Feature Combination Pooling. 9
3.2.4 k-NN Node Reduction. 11
3.2.5 Knowledge Link Determinations. 13
3.3 Training 13
3.3.1 Manual Training. 13
3.3.2 Automated Training. 16
3.4 Anomaly Scores 17
3.5 Incremental Learning 20
3.6 Accelerating AnRAD 21
3.6.1 Complexity Analysis. 21
3.6.2 Naïve Parallel Implementation. 22
3.6.3 In-memory Knowledge Bases. 23
3.6.4 Workload Balancing. 25
3.6.5 Extension to Xeon Phi Co-processor. 25
4 Results and Discussions 26
4.1 AnRAD for Traffic Monitoring of Manually Structured Data 26
4.2 AnRAD Accuracy 28
4.2.1 Abnormal Vehicle Behavior Detection. 28

4.2.2 Network Data Intrusion Applications. 30
4.3 Incremental Learning Benefits 33
4.4 Performance Evaluations 34
5 Conclusions 36
6 References 37
APPENDIX 38
LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS 39

LIST OF FIGURES

Figure 1. Zone Partitioning Algorithm 6
Figure 2. Partitioned Zones 6
Figure 3. Classification of Vehicle Records and Corresponding Lexicons 7
Figure 4. Hierarchical Lexicons Structure 9
Figure 5. Uncorrelated Components 10
Figure 6. Correlated Components 10
Figure 7. Knowledge Links among Lexicons 14
Figure 8. Neighbor Index 15
Figure 9. Confabulation Model Training and Recall 16
Figure 10. AnRAD Training and Recall Procedures 16
Figure 11. Workload Mapping 18
Figure 12. Anomaly Score Computation 19
Figure 13. Thread Pool for CPU Multi-threading 22
Figure 14. In-memory Knowledge Base Layout 24
Figure 15. Memory Usage of Individual Models 25
Figure 16. Anomaly Score of a Location Key Lexicon 26
Figure 17. Anomaly Scores of Speed Key Lexicons 27
Figure 18. Anomaly Scores of First Neighbor Pair Lexicons 27
Figure 19. Anomaly Scores of the Speed Lexicon for “Start/Stop” Events 27
Figure 20. Anomaly Scores for (a) AnRAD and (b) Bayesian 28
Figure 21. Alarm Rate v. Score Threshold of Vehicle Monitoring 29
Figure 22. DARPA Dataset ROC Analysis 31
Figure 23. ADFA-LD Dataset ROC Analysis 32
Figure 24. False Alarm Rate v. Training Time 33
Figure 25. AUC Scores v. Training Sizes 34
Figure 26. Converting ECEF Data to Geodetic Format 38

LIST OF TABLES

Table 1. Complexity Analysis 21
Table 2 Correlation between Anomalies and Outstanding Nodes 30
Table 3. Runtime Comparison Results 35

1 SUMMARY

In this project probabilistic inference-based anomaly detection was investigated, and the results were applied to wide area surveillance scenarios. The “Anomaly Recognition And Detection” (AnRAD) system, which is a cogent confabulation network, was developed and tested. This system included an abstract-level autonomous information processing framework that provided continuous monitoring and real-time anomaly detection.

Massive parallelism accelerated this inference modeling system using state-of-the-art multicore processors including Graphics Processor Units (GPUs) and Intel Xeon Phi processors. Real-time surveillance applications with high-volume input data streams were modelled with significant processing speedups.

Synthetic Aperture Radar (SAR) data from hundreds of square kilometer areas were analyzed in wide area surveillance scenario testing. In this testing road traffic was represented by a set of features that were extracted from a Ground Moving Target Indicator (GMTI) radar input stream. Likelihood-ratio testing was performed on sets of key features to determine abnormal vehicle behavior. The low-complexity learning and recall AnRAD system supported incremental learning, which enhanced the system accuracy.

The AnRAD system was also generalized for the additional application of network intrusion detection. A self-structuring technique was developed that determined a probabilistic inference network structure from unlabeled data. Without relying on data distribution assumptions, mutual information between features was leveraged to learn and produce succinct network configurations. These results were compared to several other existing anomaly detection methods.

2 INTRODUCTION

In Section 2 is presented an overview of this topic and pertinent background information.

2.1 Overview

The advanced sensing and imaging capabilities of today's Synthetic Aperture Radar (SAR) systems enable wide-area surveillance by generating large volumes of data. There is an urgent need for autonomous SAR information processing and data exploitation techniques to facilitate the continuous monitoring and prompt anomaly detection of this data that may be collected from areas of hundreds of square kilometers.

Confabulation theory offers a comprehensive detailed explanation of the mechanism of thought in humans and other vertebrates. In terms of computational reasoning, the term confabulation is also used to describe inductive reasoning via Bayesian networks. In this project, we applied the Cogent Confabulation model [1] to develop an abstract-level autonomous information processing framework that performed real-time anomaly recognition and detection.

Anomaly detection refers to techniques that identify patterns in data that do not conform to the expected observations. This project focused on developing an abstract-level autonomous information framework that would provide continuous monitoring of vehicle behaviors over a very large area using unsupervised machine learning. Taking advantage of the advanced sensing and imaging capabilities of today's SAR systems, our framework focused on anomalous traffic situation detection for wide area surveillance.

Several factors made building such a system challenging. It is very difficult to exactly define what the "expected" vehicle behavior is. Once that is determined, then behavior that can be differentiated from that is defined as abnormal vehicles behavior. Secondly, normal traffic situations may evolve over time, so it is critical that a system is adaptive to such changes. Thirdly, modeling the traffic behavior and designing the learning algorithm can be complex since there may be thousands of vehicles within an area of interest. Last but not least, the requirements to handle such a large volume of inputs and provide real-time monitoring impose stringent computation performance requirements.

Several anomaly detection techniques [2] have been developed, including a kernelized one-class Support Vector Machine (SVM) classifier [3]. Replicator neural networks [4] also provide anomaly determinations by a measuring the "outlierness" of data. Nearest neighbor approaches [5] assume that normal data occurs in dense neighborhoods, and that anomalies are more distant from their neighbors. This has been improved by Huang [6] using a rank-based approach. Bayesian network-based methods have been proposed by many researchers, including Sebyala [7], as the basis for anomaly detection. Fu [8] determined traffic anomalies from camera imagery by exploiting trajectory clustering, while Shen [9] developed a Gaussian mixture model. These approaches neither systematically resolved the problem of monitoring very large areas nor did they exploit the relationship between adjacent vehicles.

To solve these additional problems, here we developed and tested the autonomous Anomaly Recognition And Detection (AnRAD) framework. This framework is based on cogent confabulation [1], which is a probabilistic inference model that mimics human information processing. It extracts the conditional probability among symbolic representations of features in

an unsupervised environment. A large surveillance area is first partitioned into smaller zones that are then independently processed. A Knowledge Base (KB) is built for each zone by feeding traffic records into the properly modeled knowledge networks. When new traffic information is received, anomaly scores are then calculated by means of the likelihood-ratio test for observed events. Events with high anomaly scores are then marked as potential anomalies, and alarms are sent to a human observer. The AnRAD requirements for the unique features of this platform are summarized as follows:

1. The AnRAD system must be able handle a large volume of vehicle traces over large areas. The surveillance area will be partitioned that is based on traffic density information extracted from the training data. In this way, the computation load will be balanced, and the inference model can be constructed more accurately.

2. The confabulation-based model had low complexity for both training and recall. Therefore, the ANRAD system must even be trainable during operating time, which will enable continuous improvement to the Knowledge Base (KB) quality.

3. The AnRAD system must be capable of capturing contextual information among vehicles and their neighbors. Abnormal events such as tailgating that is caused by interactions among vehicles must be detectable. Such events have not been considered in previous research.

4. The overall system must have a hierarchical architecture so that the work load in each level of the hierarchy is inherently parallel. A Graphics Processing Unit (GPU)-based parallel implementation will be developed to achieve computation acceleration of the AnRAD system.

2.2 Confabulation-based Network Architecture

The network architecture for confabulation-based anomaly detection is by definition specific for each application. The neuron nodes (i.e. lexicons) and the synapses (i.e. knowledge links) between them must be re-configured when applied to different applications. To build such networks requires expert knowledge. This limitation makes the same confabulation-based anomaly detection architecture inflexible in handling different datasets. To address this, a self-structuring technique needed to be developed that would enable the automatic construction of the confabulation architectural network. It must concretely learn from the data a succinct set of nodes that represent the original features or combinations of the features.

Each node is to be associated with a lexicon, which will record the symbolic representations of the possible inputs. The links between the nodes must also learn from the initial data. Given the learned network configuration, further incoming data streams must then incrementally refine the weights of the knowledge links. These weights will be the conditional probabilities between the lexicon symbols. With this self-structuring technique the AnRAD framework will be generalizable to a wider range of applications. In addition to road-traffic monitoring that was the primary application for this effort, the AnRAD system was also applied to a network intruder detection application.

2.3 Cogent Confabulation

Cogent confabulation is a bio-inspired computational model that mimics human information processing. The AnRAD system is based on a cogent confabulation model that performs probabilistic inferences. An observation is represented as a set of features. From these features the basic dimensions are constructed that describe the specific application, e.g. vehicle speed and

position coordinates. The observed attributes of a feature are referred to as symbols. A set of symbols that describes the same feature forms a lexicon. The symbols in a lexicon are mutually exclusive.

Knowledge Links (KLs) are established among the lexicons. The KLs are directed edges that extend from the source lexicons to the target lexicons. Each KL is associated with a matrix. The *ij*th entry of the matrix gives the conditional probability $\log[p(s_i|t_j)]$between the symbol s_i in the source lexicon and the symbol t_j in the target lexicon. The knowledge matrix is constructed during training by extracting and associating features that were extracted from the training data.

The cogent confabulation model has a close resemblance to a neural system. The symbols are analogous to neurons, and KLs between the symbols are analogous to synapses between neurons. Whenever an attribute is observed, the corresponding symbol (i.e. neuron) is activated, and an excitation is passed to the other symbol(s)[1] through the KLs[2].

The excitation level (*el*) of a symbol *t* in lexicon *l* is calculated by summing up all of the incoming KLs:

$$el(t) = \sum_{k \in F_l} \left(\sum_{s \in S_k} I(s) \ln \left(\frac{p(s|t)}{p_0} \right) + B \right), \quad (1)$$

where F_l denotes the set of lexicons that has connections to l, and S_k is a set that consists of collections of symbols in lexicon k; $I(s)$ is the firing strength of source symbol s, and it is set to 1 if s is observed without ambiguity; p_0 is the minimum probability that is considered informative. Parameter B is a constant called the band gap. It is 0 if none of the active source symbols in S_k has KLs that are connected into t. The band gap ensures that the symbols with more KLs receive higher excitation over those with fewer KLs.

The excitation level of a symbol is actually its log-likelihood given the observed attributes in other lexicons. The excitation levels can be used to resolve the observation ambiguities via the maximum likelihood inference [10]. A confabulation model [11] has been applied to text recognition, which demonstrates its ability to handle incomplete data and capture causal relationships between observations. Here the excitation level enables the anomaly detection by using the likelihood ratio test.

When compared to other schemes, the training and recall processes in the confabulation model are simplified. Also, since the model is highly configurable, the system is easily modifiable, which is a requirement of the system. It is also easily optimized for improved performance. Finally, because training and recall share the same knowledge data structures, the model offers unsupervised learning and online updating capabilities.

[1] i.e. neurons
[2] i.e. synapses

3 METHODS, ASSUMPTIONS, AND PROCEDURES

The basic AnRAD system for wide area surveillance was designed with four functional areas: zone partitioning, confabulation networks, training, and detection. Network self-structuring enhanced the flexibility of the AnRAD framework and increased its applicability. The system was accelerated through high performance computing parallel processing.

3.1 Zone Partition

The test SAR surveillance data were typically collected from hundreds of square kilometer areas, and within these surveillance areas were observed thousands of vehicles. To provide real-time learning and detection, an area was partitioned into multiple zones, and parallel processing was then performed on each of the zones. This was designed in part to reduce the complexity of learning and detection. Zone partitioning also resulted in the following benefits:

1. More accurate modeling that captured local details. The traffic situation varied from location to location. Therefore it was not appropriate to describe the whole area by a single model, since vehicles exhibited different behaviors in different zones. Zone partitioning helped to improve the accuracy of the model.
2. Reduced model complexity and memory requirement. The number of possible attributes of certain features, e.g. vehicle coordinates, was directly proportional to the size of the monitoring area. Therefore zone partitioning effectively reduced the number of symbols in a lexicon, and this reduced the complexity of the confabulation model.
3. Enabled parallel processing. Each detection zone was trained and operated as an independent unit. The workload was easily parallelized and the knowledge base could be stored in a distributed manner.

The number of vehicles in a partitioned zone determined the computation workload of the training and recall processes. The partitioning algorithm that was developed is shown in Figure 1. This algorithm divided the area into zones that were based on the average traffic density, and ensured that none of the detection zones had more than N vehicles appearing in the same time slot (frame) in the training set. The resulting zones were organized as a sibling tree structure. Each parent node was associated with four child zones, which were derived by splitting the parent zone into four equally-sized patches.

```
1  Find the MAX and MIN values of vehicle coordinates
2  Define the root zone and set Z = {root}
3  For each time slot t in the training set
4     reset vehicle_count of each zone in Z
5     For each record x at time slot t
6        Find zone z that x.location falls into
7        if z.vehicle_count exceed limit N
8           Split z into 4 child zones z[1-4]
9           Update vehicle_count of z[1-4]
10          Update set Z
11       Else
12          z.vehicle_count++
13       End if
14    End for
15 End for
```

Figure 1. Zone Partitioning Algorithm

An example of the resulting zone partitioning is shown in Figure 2. Each zone was designed to have a "buffer area", which are the margins of a zone that overlap with its adjacent zones. Vehicles in these buffer areas were not tested for anomaly detection, but were used to generate supporting information, such as the neighbors to the target vehicles and the previous records of the current target vehicles. This allowed the system to also consider vehicles about to enter or that have already exited a zone.

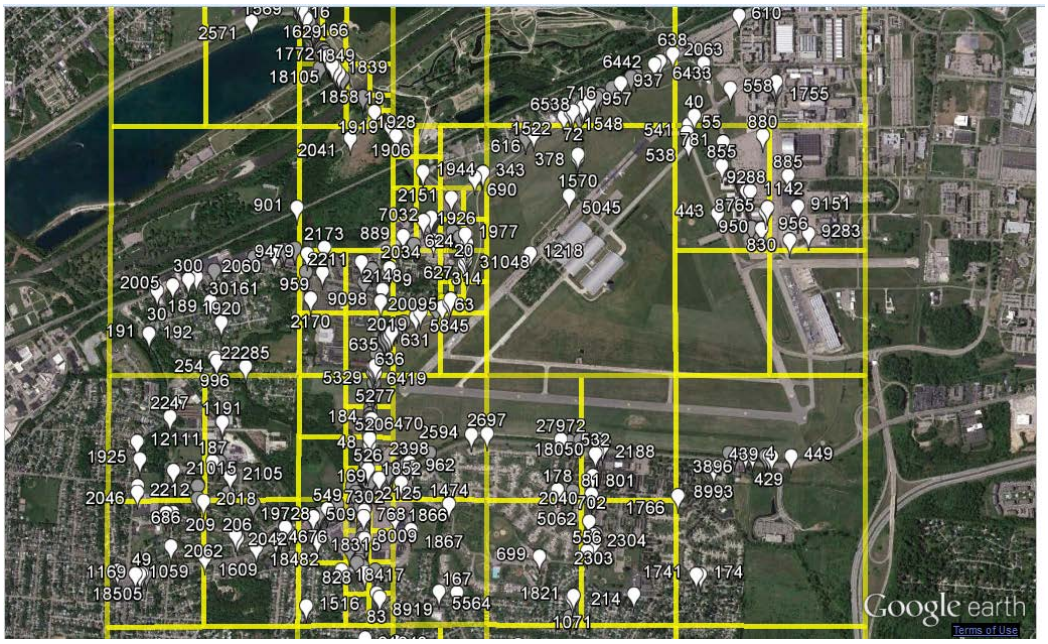

Figure 2. Partitioned Zones

3.2 Confabulation Networks

A confabulation model was manually built for each of the partitioned zones. The first task was to define the lexicons, which are features that were used to describe the vehicle behavior in traffic. In the anomaly detection problem, the behavior of a vehicle was considered within the context of its neighbors during the current and previous observations. When all of the observations were made in the same time slot as a frame, the detection involved three consecutive frames. Four classes of objects were then observed in a scene: Target, Neighbor, Auxiliary Center, and Supporter.

Each vehicle that appeared in a detection zone at frame *t* was considered a *target*. The ten vehicles closest to the target in the same frame were defined as *neighbors*. Based on the current location and speed of target, its location was estimated in the previous frames. The estimated targets in frames *t-1* and *t-2* were referred as the *auxiliary centers*. The nearest ten neighbors of the auxiliary center in the corresponding frame were called the *supporters*. In Figure 3 is shown an example of the four types of vehicle records. A scene was generated for each target observation within the context of neighbors, auxiliary centers and supporters.

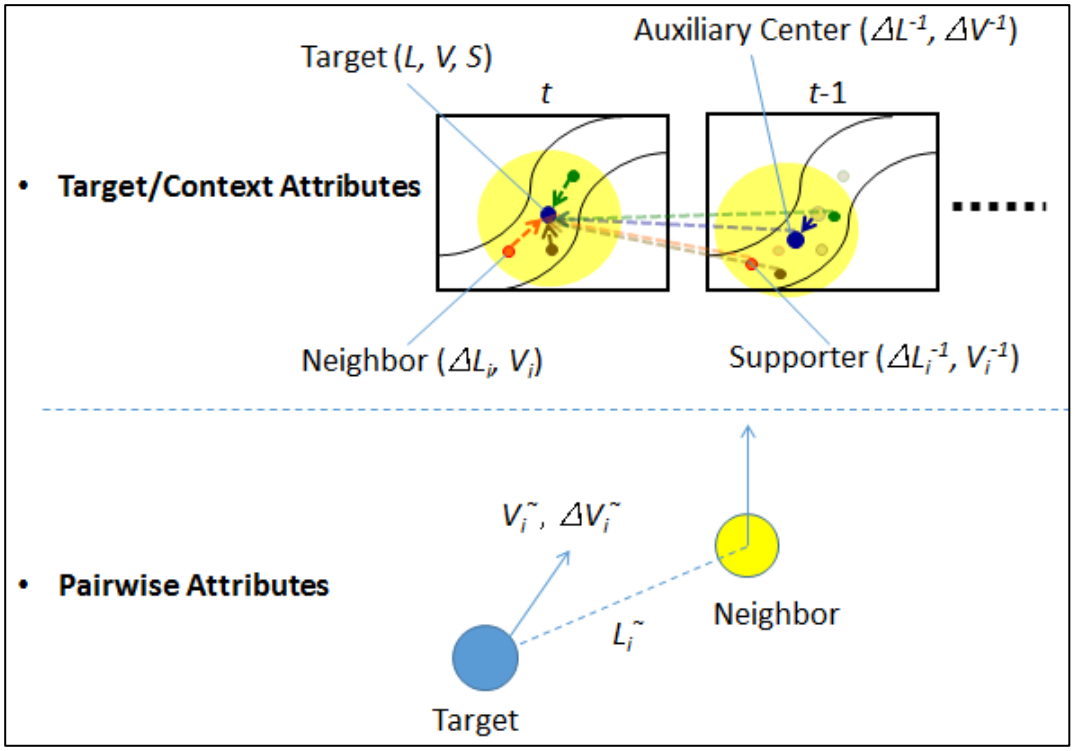

Figure 3. Classification of Vehicle Records and Corresponding Lexicons

Three lexicons were used to describe the basic attributes of a target vehicle: Target Location (*L*), Target Velocity (*V*), and Target Size (*S*). The target location was expressed in geographic latitude and longitude and was discretized to levels of approximately ten meters. The target speed was expressed as the combination of the ground speed and the direction. The target size was discretized to reflect the five different vehicle categories:

1. Sedan,
2. Truck,
3. SUV,
4. Moving Truck, and
5. 18-wheeler Truck.

3.2.1 Confabulation Network Self-structuring.

Our experimental results showed that the AnRAD was a promising approach to detect anomalous streaming data, but its performance essentially depended on the selection of key lexicons and their incoming links. To construct the confabulation network required application specific knowledge. Expert knowledge will not always be available, and it does not always ensure optimal network structure. Self-structuring then plays an important role to improve the framework's generality and applicability.

In the AnRAD framework, inputs were represented as *N* streams $\{\{x_1^1, x_1^2, \dots, x_1^t, \cdots\}, \{\cdots, x_2^t, \cdots\}, \cdots, \{\cdots, x_N^t, \cdots\}\}$ generated from the certain distribution *D*. Here x_n^t represented a record tuple of the *n*th stream at time frame *t*. It consisted of *Q* features denoted by $x_n^t(q)$. In the self-structuring stage, a span of the data at frame $[0, T_g]$ was sampled and used to construct the confabulation network *G* that best described the application. After the network was constructed, new streams at $(T_g, T_0]$ were used to train the initial knowledge base $KB_G^{T_g:T_0}$. The KB was applied to streams at $(T_0, \cdots)$ to generate network anomaly scores for each frame. At the same time, the new incoming data continuously refined the knowledge base $KB_G^{T_g:t}$. Typically, a moving window with size *W*, $\{x_n^{t-W}, \cdots, x_n^{t-1}, x_n^t\}$ was applied to the input stream at frame *t* to form the scene. In the next subsection the generation of network *G* and refinement of knowledge base $KB_G^{T_g:t}$ are discussed.

3.2.2 Key Node Hierarchy.

The confabulation model captured the first order relationships between features. Higher order relations were considered by adding new lexicons corresponding to the feature combinations. The final structure of confabulation network consisted of hierarchical lexicons that where higher level nodes that were formed as the compositions of lower-level nodes, as shown in Figure 4. Lexicons at the bottom layer represented single primary features. These primary features provided a basic description of the input data. The higher level lexicons assembled multiple primary features, and represented more abstract meanings and combinational patterns. The layered structure provided direct mapping from the feature space to nodes in the knowledge graph, but its complexity increased exponentially with naive implementation. Since the confabulation network worked at the symbolic level, continuous features were discretized before mapping to symbols in the lexicons. Given the composition of features, higher-level lexicons had coarser discretization intervals than the lower level lexicons.

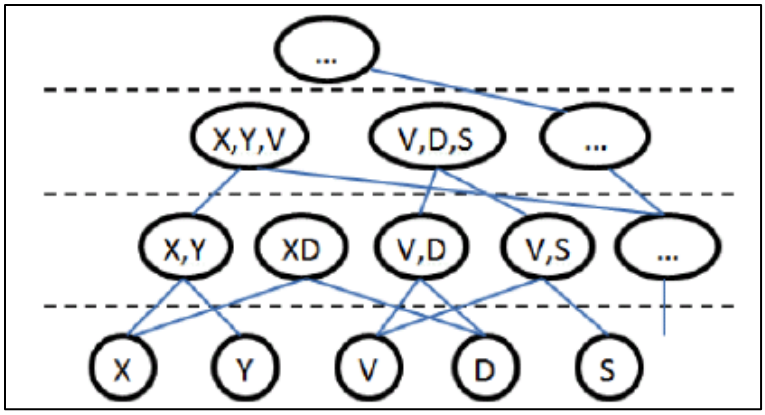

Figure 4. Hierarchical Lexicons Structure

Note that each input data point was a segment of the data stream within a time span. The primary features may also have had a timestamp. The composition of features were not just spatial, but also temporal. For example, there may have been a feature composition:

$$\langle x_n^t(q), x_n^t(q')\rangle,\ q, q' \in Q, \text{ or } \langle x_n^t(q), x_n^{t-\Delta t}(q)\rangle \quad (2)$$

$$\Delta t < W. \quad (3)$$

The temporal relations among them were learned and checked.

A question raised here was which feature combinations should be included. If the model simply considered all of the possibilities, it would quickly scale to an intractable size as *Q* and *W* increased. This is both unnecessary and computationally wasteful. Another option was to rely on traditional feature reduction techniques, such as principal component analysis. Although these techniques have long been studied, they either required supervised learning or destroyed the direct relation (one-to-one mapping) between feature space and lexicon space. Furthermore, none of the previous techniques have been applied to select feature combinations. A *pooling and reduction* procedure was used here that was applied to both spatial and temporal domains to construct the key lexicon's hierarchy. A link selection algorithm was also developed to establish the knowledge links between the lexicons.

3.2.3 Feature Combination Pooling.

Feature pooling is when primary features are complemented with a set of composite features, and this captures the higher order relations. The pooling stage generates a set of lexicon candidates, which can then be reduced, as discussed in the next section.

For a simple two-feature combination; the first question asked is whether such a combination will provide more information for anomaly detection than the individual feature components. Consider the example scatter plots in Figures 5 and 6, where the *X* and *Y* axes represent the dimensions of the two primary features in arbitrary units.

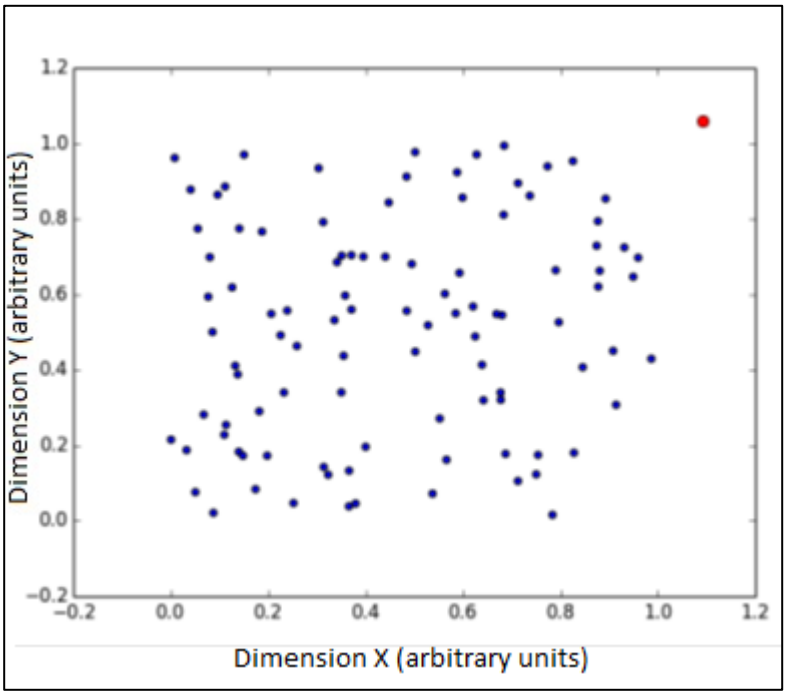

Figure 5. Uncorrelated Components

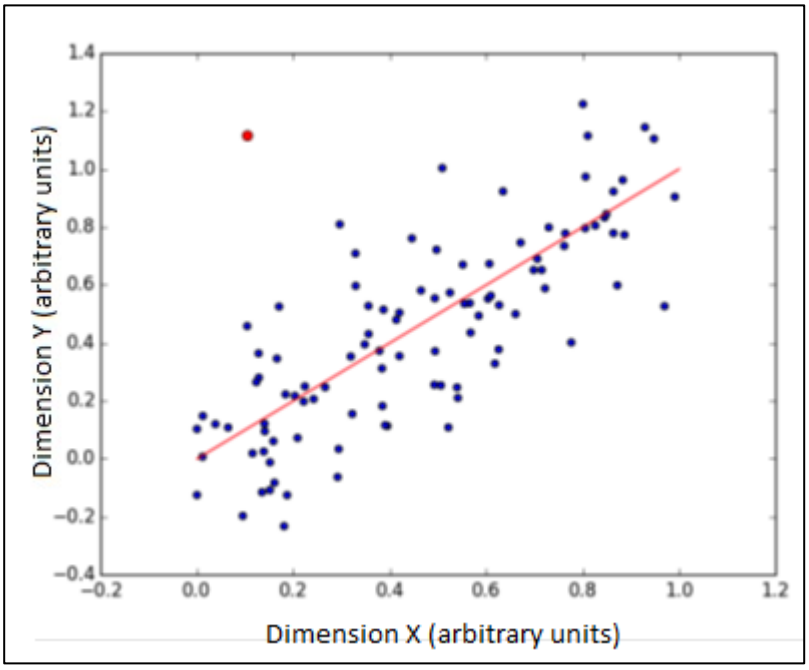

Figure 6. Correlated Components

If the two features are distributed independently in their feature space, as are the blue dots in Figure 5 a potential outlier (the red dot) in this subspace is detected by considering only one of

the components. Therefore the combinations of non-correlated features do not offer additional information. However, if the two features are sufficiently relevant, as shown in Figure 6, the red dot, which is originally indistinguishable from any single axis, is detected by their combination. Based on this observation, the pooling procedure was designed to be a combination of highly correlated features.

To extend this concept to more general cases, the feature distance was defined as $d(q_i, q_j) \in [0, 1]$. The shorter the distance the more relevant were the two features q_i and q_j. Since the framework had to be applicable to both continuous and categorical data, the normalized mutual information was adopted as the distance measure. For combination Q_l consisting of two or more features, a simple relevancy test was performed to determine whether it should be included in the lexicon candidate set:

$$RT(Q_l) = \prod_{q_i, q_j \in Q_l} I\left[d(q_i, q_j) < d_{prox}\right] \quad (4)$$

This test required that all of the component pairs in Q_l were within a constant proximity distance d_{prox} to each other. Algorithm 1 pooled the features for the lexicon generation. The algorithm first added all of the single features into the candidate set. Then in the second for-loop each subset of Q whose cardinality was less than *max_order* was inspected. If the subset passed the relevancy test, a new lexicon candidate was added to CS. Not all of the candidates were key lexicons whose anomaly score was calculated. A reduction stage was also used to select the key lexicons from the candidate set.

Algorithm 1 Feature Combination Pooling

```
procedure pool(Q, max_order):        # Q: the complete feature set; max_order: the maximum combination order
CS ← empty set
for each feature q ∈ Q:
    add {q} to CS
for each Q_l ⊂ Q and |Q_l| < max_order:
    if all Q_l' ⊂ Q_l was accepted and RT(Q_l) passed:
        add Q_l to CS
return CS        # feature combination candidate set
```

3.2.4 k-NN Node Reduction.

Although the pooling process excluded most of the irrelevant combinations, the number of possible candidates was still large when Q had many features. Therefore, a reduction procedure was used to further compress the candidate set that generated the key lexicons. The redundancy among candidates selected in the pooling stage was removed during this reduction. Because labels were not available in the training set, a similarity-based method [12] was modified to preserve the most representative combinations.

Algorithm 2 Node Selection

procedure knn(CS, K): # CS: pooled candidate set; K: the initial K
KEY $\leftarrow CS$
for each combination $Q_l \in CS$:
    for $Q_k \in K$ nearest neighbor of Q_l:
        Q_l.dist[k] $\leftarrow d(Q_l, Q_k)$
find Q_0 who has the smallest k-distance
max_err $\leftarrow Q_0$.dist[K-1]
while $K > 1$:
    find Q_r who has the smallest k-distance
    remove Q_r 's K nearest neighbor from KEY
    radius $\leftarrow \min(Q_x.\text{dist}[K-1],\ Q_x \in \text{KEY})$
    while radius > max_err:
        $K \leftarrow K - 1$
        radius $\leftarrow \min(Q_x.\text{dist}[K-1],\ Q_x \in \text{KEY})$
return KEY # key node set

The general idea of the reduction procedure was to cluster the candidate feature combinations by similarities, and then select one representative from each of the clusters. Again, normalized mutual information was employed to measure the distance $d(Q_{11}, Q_{12})$ between the combinations. The clustering process was accomplished by *k*-NN (k nearest neighbor) principle. When the most compact candidate was selected from a cluster, its neighbors were discarded. This operation was repeated until the remaining candidates no longer formed a cluster. The reduction procedure is described in Algorithm 2.

The algorithm first initialized the set KEY with all of the candidates. Then it calculated the distances for each combination to its nearest neighbors. The center of the compact cluster's *k*-distance was selected as the upper limit of the cluster radius. Then in the following while-loop, the combination with the minimum *k*-distance was selected, and its *K* neighbors were removed from the KEY set. Then the *K* value was reduced until the next cluster's radius was shorter than the radius limit. The neighbor-removing process repeated until *K* reached 1. The remaining candidates in KEY set were the final key lexicons selected.

Although the features in Q were used as an example to explain the pooling-reduction procedure, the concept can also be applied to temporal domain. When the data inputs were not single points in the feature space, but multi-variant time series, the definition of anomalies can be extended to historical patterns. To capture such potential outliers, the key lexicons must include not only the different features, but also the feature projections in different frames. This can be accomplished by performing feature-wise selection followed by temporal selection.

If multiple frames were considered after the key lexicons were determined, then each lexicon along with its historical readings formed a new temporal feature set $Q_l^W = \{Q_l^0, Q_l^{-1}, \cdots, Q_l^{-W}\}$. The same pooling-reduction algorithms were then directly applied on these feature sets to generate informative and succinct key lexicons. A key lexicon was represented as a two-dimensional pattern:

$$R_l \sim \left[(q_{l_1}, q_{l_2}, \cdots, q_{l_i}, \cdots)^{-t_1},\ (\cdots, q_{l_i}, \cdots)^{-t_2}, \cdots, (\cdots, q_{l_i}, \cdots)^{-t_j}, \cdots\right] \quad (5)$$

The number of correlated frames *W* was usually much smaller than the feature number in *Q*, so that the reduction process was sometimes omitted in the temporal selection.

3.2.5 Knowledge Link Determinations.

Now that the key lexicons were identified, the next step was to find the supporting lexicons that could be used to infer the key lexicon symbols. To do so a max-similarity, min-redundancy principle was followed. For instance, to infer the shape of an object, touching was preferable compared to color (max-similarity). But if touching was already selected, weighting might not be necessary as they share some information (min-redundancy). Generally, we wanted to maximize the correlation between key lexicons and their supporting lexicons, and meanwhile, minimize the correlation among the supporting lexicons that were connected to the same key lexicon. The supporting lexicons were chosen from the primary features since the key lexicons had already handled the combinational patterns.

Algorithm 3 Link Selection

```
procedure select_links(R, F):        # R: target node; F: set of single features
SUPP ← empty set
ranks ← F.sort(key=d(R, q ∈ F))
for feature q ∈ ranks:
    if q ∈ R or d(R, q) > (1 − d_prox):
        continue                     # low similarity
    if any p ∈ SUPP has d(p, q) < d_prox:
        continue                     # high redundancy
    add q to SUPP
return SUPP                          # support nodes for R
```

Heuristic Algorithm 3 defined a group of features at certain time offset, $\{q^{-t}, q \in Q, t < W\}$ which inferred the observation at a key lexicon R_l. The algorithm first sorted the supporting features by their distances to the target key lexicon. Then it traversed the sorted features, added a primary feature to the supporter set only when: (1) it was not one of the components of the key lexicon; (2) it was highly correlated with the key lexicon; and (3) it had a low correlation with supporting features that had already been selected for the same key lexicon. At this point the confabulation model was properly configured, and the training streams were then processed to build the KB.

3.3 Training

3.3.1 Manual Training.

Two lexicons were associated with each auxiliary center, the center displacement (ΔL^{-t}) and the center acceleration (ΔV^{t}), at frame t = 1, 2. The center displacement was the distance between the target and the auxiliary center, and was represented by the displacement in latitude and longitude. It described how far the target moved in the last 1 and 2 frames. The center acceleration specified the change of velocity (speed and direction) of the target during the last 1 and 2 frames.

Two lexicons were associated with each neighbor or supporter. The relative location lexicon[3] gave the relative position of the neighbor (or supporter) with the respect to the target. The velocity lexicon (denoted as V_i for the ith neighbor, and V_i^{-t} for the ith supporter in frame -t) specified the velocity of the neighbor (or supporter) as a combination of the speed and direction.

Three lexicons were used for pairwise attributes that described the relation between the target and each of its neighbors. Pairwise location lexicon ($L_i\tilde{}$) specified the distance (in meters) and direction (in degrees, relative to neighbor's moving direction) between the target and the ith neighbor. Pairwise speed lexicon ($V_i\tilde{}$) specified the target's absolute speed (in m/s) and relative direction (in degrees) with respect to the neighbor's direction. Pairwise speed change lexicons ($\Delta V_i\tilde{}$) captured the target relative speed (in m/s) and relative direction (in degrees) with respect to the ith neighbor.

In total 100 lexicons (i.e. features) were used to describe the status and context of a target vehicle. The set of observed attributes of these features formed a sentence, which was the basic input for the confabulation training and recall processes. Every vehicle in the detection zone was treated as a target; and a scene was created for each one of them.

In Figure 7 is shown the overall confabulation model with lexicons and the knowledge links that connected them. Lexicons S, L, V and $L_i\tilde{}$, where $1 \leqslant i \leqslant 10$, were represented using dashed circles. Each one of them corresponds to a general category of abnormal behavior of the target vehicle, such as abnormal location or speeding, inconsistency between vehicle size and its status, and abnormal interactions with neighbors. We refer these lexicons as the key lexicons and others as the regular lexicons. Only the excitation levels of the key lexicons were evaluated. All other lexicons provided supporting context for them. A key lexicon has incoming knowledge links from all of the other lexicons, while a regular lexicon has outgoing knowledge links.

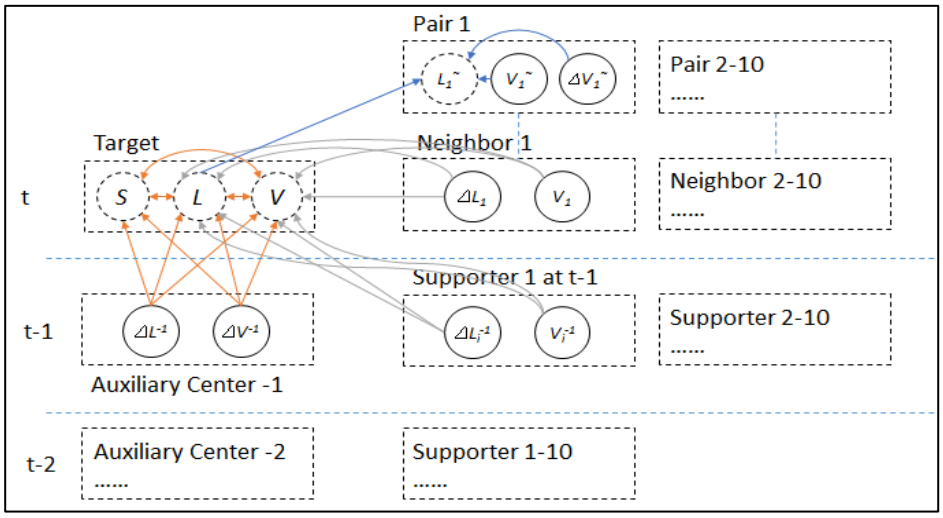

Figure 7. Knowledge Links among Lexicons

[3] denoted as ΔL_i for the ith neighbor, and ΔL_i^{-t} for the ith supporter in frame -t

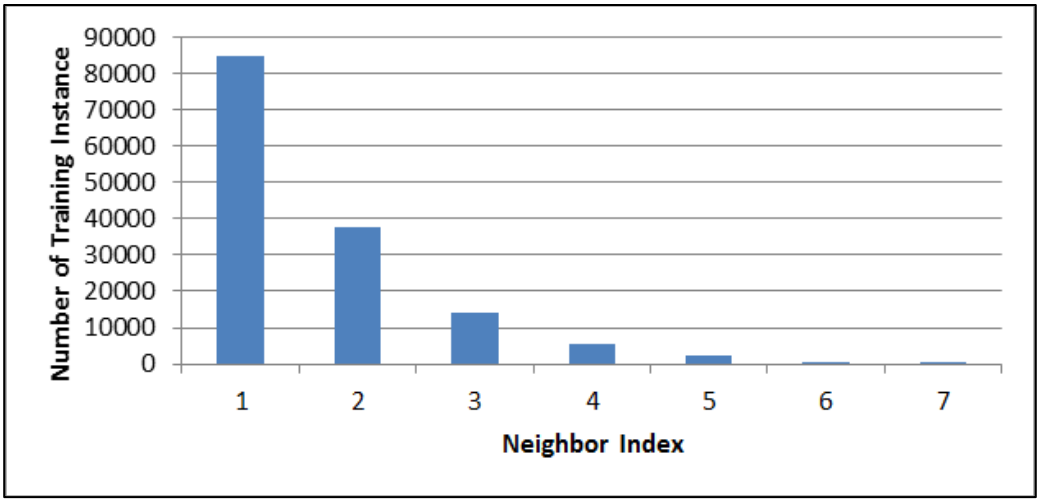

Figure 8. Neighbor Index

In previous confabulation models [1] [11] ten neighbor lexicons were necessary. In Figure 8 is presented the neighbor index that was determined for our data. A target vehicle only had a small number of neighbors, which was usually less than five. Therefore, the neighbor lexicons with the most indices had the smallest training data. Because of this, this work developed a new technique called “shared links”

The amount of available training data directly affected the accuracy of the model. Meanwhile neighboring vehicles with similar behavior were associated more often during multiple times of observation, because the neighbors were mapped to lexicon indices-based on their relative distance to the target. Our shared links model overcame these problems by letting the knowledge links initiated from the neighbor lexicons share the same knowledge matrix. In other words, we did not distinguish neighbors. In this way the training data for the neighbors all contributed to the same KB. This also reduced the false alarm rate. Moreover, the shared link knowledge helped to reduce the size of KB, since one matrix was maintained for multiple knowledge links.

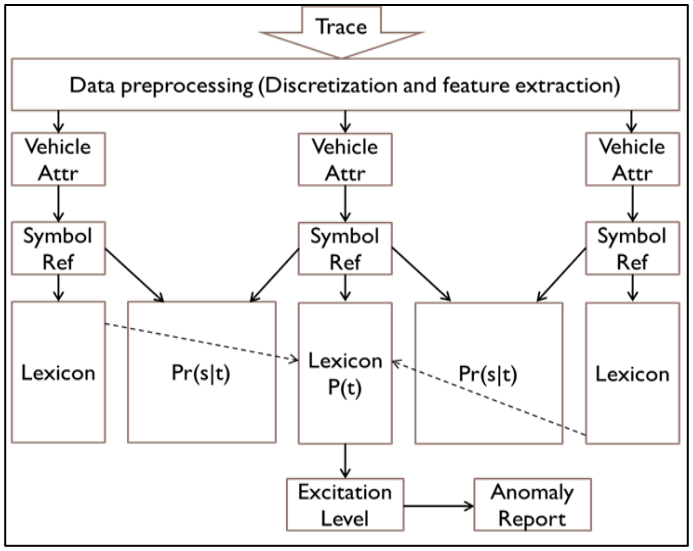

Figure 9. Confabulation Model Training and Recall

3.3.2 Automated Training.

The automated confabulation algorithm consisted of two procedures: unsupervised learning (*training*) and anomaly detection (*recall*). Both procedures operated on the same KB. In Figure 9 is shown the task flow diagram of the training and recall procedures. The observed traffic data were first preprocessed by zone partitioning and feature extraction. The observed lexicon attributes were collected and assembled into a scene. Each observed attribute was mapped into a globally unique reference number called a *symbol* using two-level hash functions. For a given lexicon, all of the observed attributes during the training process formed the candidate set. After preprocessing each knowledge link, the co-occurrences of the source and target symbols were counted, and the log-conditional probabilities were calculated. Lastly, the knowledge matrices were stored as KBs. These steps are summarized in the Training Section in Figure 10.

Training	**Recall**
1 Reset all KLs and lexicons	1 **For each** input sentence *T*
2 **For each** input sentence *T*	2 Map elements in *T* into reference numbers
3 Map elements in *T* into reference numbers	3 **For each** target key lexicon *l* and observation *t*
4 **For each** symbol o_i in *T*	4 Calculate *el*(*t*) by Eq. (1)
5 Add o_i to the candidate set of lexicon[*i*]	5 **For each** candidate's t_j in lexicon[*l*], find $el(t_{best})$
6 **For each** symbol o_j in *T*	6 **End for**
7 If (*i* != *j*), KL[o_i, o_j].count++	7 Calculate *nas*() by Eq. (2) (3)
8 **End for**	8 If bucket overflows, flag anomaly on the target
9 **End for**	9 **End for**
10 **End for**	10 Output anomaly reports
11 Finalize value of each KL and store into KB	

Figure 10. AnRAD Training and Recall Procedures

The same preprocessing and feature extraction training procedures were performed during a recall. The excitation level *el* of each key lexicon was calculated, which gave the likelihood value of the observation given the context of the target and neighbors. For each key lexicon the excitation levels of other symbols in its candidate set were also calculated, and the symbol $t_{best}$with the highest excitation level was obtained. An *anomaly score as*(*l*,*t*) was defined based on the *likelihood ratio test* as shown in Equation (6).

$$as(l,t) = \frac{el(t_{best}) - el(t)}{el(t_{best})} \tag{6}$$

A high anomaly score for an observed symbol *t* indicated that the likelihood of *t* was much lower than the likelihood of a typical observation. The score reflected how low the observed symbol's cogency was under the given context. The anomaly score of all key lexicons were then weighted by symbol's prior probability, which was then consolidated into a network anomaly score $nas(v_{l=1\cdots L})$ that was calculated as the following:

$$nas(v_{l=1\cdots L}) = \frac{\sum_{l=1}^{L} as_l^*(v_l)}{L}, \tag{7}$$

$$as_l^*(v \in S_l) = \frac{[el(t_{best}) + pr(t_{best})] - [el(v) + pr(v)]}{el(t_{best}) + pr(t_{best})}, \tag{8}$$

where $pr(v)$ was the log prior probability of symbol *v*, *L* was the number of key lexicons, and the output score was within the range [0, 1].

3.4 Anomaly Scores

The AnRAD framework and the baseline methods generated anomaly scores for each input frame. A leaky bucket algorithm [13] with a capacity 2.0 and leak rate of 0.5 was used as the decision stage for the data streams. Whenever the score generated by the above method exceeded its threshold, it filled 1 unit into the bucket. The system reported anomalous when the water overflowed the bucket.

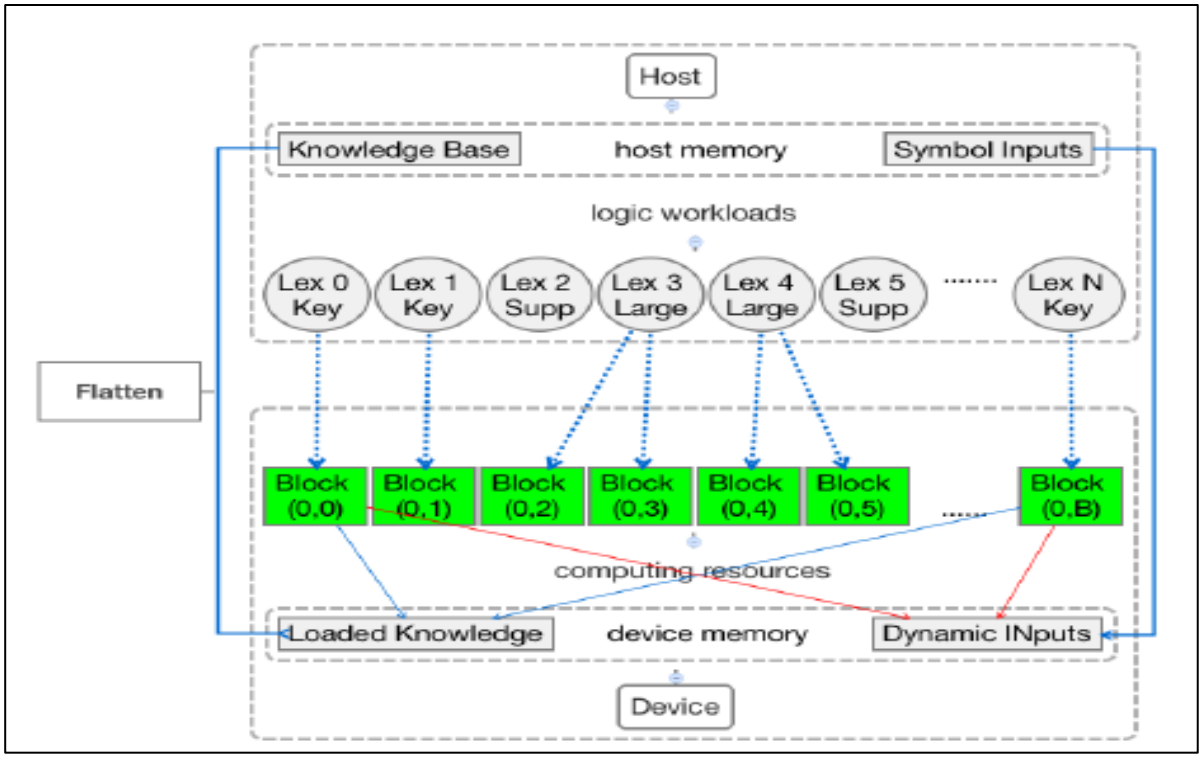

Figure 11. Workload Mapping

The kernel for computing the anomaly score was divided into two stages, the excitation mapping and the score reduction, as shown in Figure 11. The mapping stage calculated the excitations of the symbols for a key lexicon, and stored them in the shared memory buffer. Consider a key lexicon R_l with symbol set S_l and supporting lexicons $\{R_k | k \in F_l\}$.

When the kernel received a new input, it used the input supporting symbols $\{t | t \in S_k\}$ to locate the activated strips from the knowledge link LIL. Such a strip contained all the conditional probabilities for one supporting symbol *t*, $\{p(s|t) | s \in S_l\}$. At this point, if a thread was to locate a specific $p(s|t)$, the entire strip would need to be searched, which would have degraded the cache performance. Thus, a reversed approach was used in which the threads read consecutive values from the strips and accumulated these values atomically to the key-symbol buffer. To prevent control divergence, the strip-lengths were warp aligned so that the threads of a warp follow the same control flow. The cache performance was optimized since the strip access patterns were continuous.

The same preprocessing and feature extraction training procedures were performed during a recall. The excitation level *el* of each key lexicon was calculated, which gave the likelihood value of the observation given the context of the target and neighbors. For each key lexicon the excitation levels of other symbols in its candidate set were also calculated, and the symbol $t_{best}$with the highest excitation level was obtained. An *anomaly score as*(*l*,*t*) was defined based on the *likelihood ratio test* as shown in Equation (6).

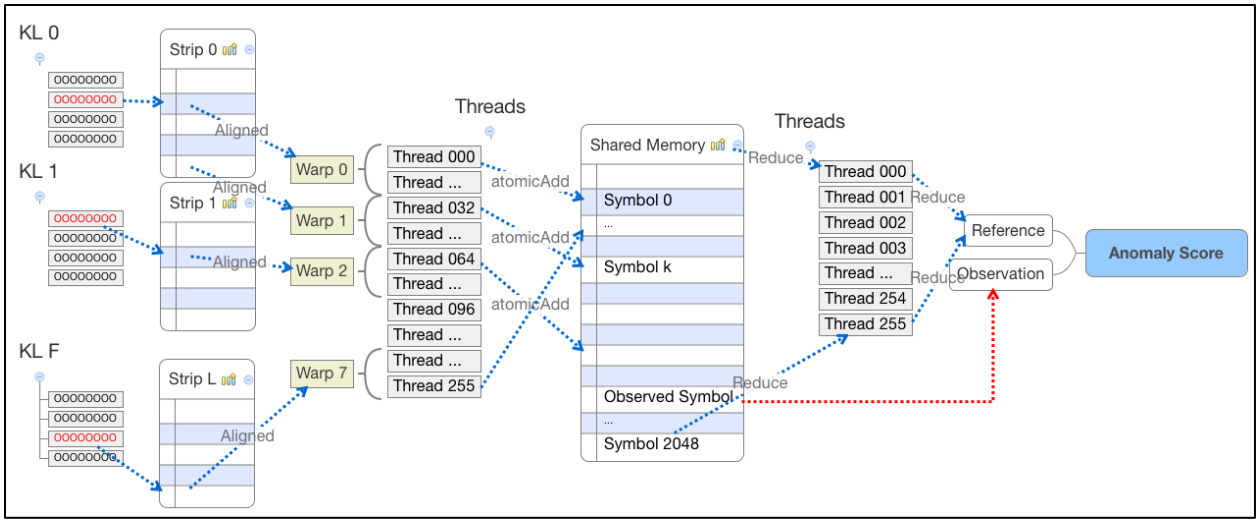

Figure 12. Anomaly Score Computation

The excitations of the key symbols S_l were stored in the shared memory for efficient atomic addition and inter-thread operations. If a lexicon had more symbols than the pre-defined shared memory usage of U_{max} > 2048, then it was partitioned into multiple blocks, as shown in Figure 12. A block dimension of 256 was jointly chosen based on the U_{max} for full occupancy. The atomic addition did not cause performance degradation: when a lexicon had many symbols, the possibility that multiple threads would write the same symbol was low; if the lexicon had very few symbols, the computation of this lexicon itself was less time consuming, and hence didn't limit the overall performance.

The same preprocessing and feature extraction training procedures were performed during a recall. The excitation level *el* of each key lexicon was calculated, which gave the likelihood value of the observation given the context of the target and neighbors. For each key lexicon the excitation levels of other symbols in its candidate set were also calculated, and the symbol $t_{best}$with the highest excitation level was obtained. An *anomaly score as*(*l*,*t*) was defined based on the *likelihood ratio test* as shown in Equation (6).

In the reduction stage, the excitations of the observed key symbol were stored first. Then all the excitations buffered in the shared memory were compared and reduced to find the most likely symbol $ref_{t \in S_l}$. The anomaly score was calculated using the excitation of this reference symbol and that of the real input symbol based on Equation (2). In lightly loaded applications, the node scores from blocks was transferred back to the host for calculating the network anomaly score. Alternatively, another kernel was launched to further reduce the scores across key lexicons when the model was more complicated or there were several concurrent input streams. The formal representation of the reduction process is shown in Algorithm 5.

Algorithm 5 Optimized Recall Kernel

```
1: procedure optimized_recall(KB, IN):        # KB:
   knowledge base; IN: input symbols
2: shared memory: exbuf[U_max]
3: block ← KB.blocks[blockIdx.y];
4: N ← block.num_symbols
5: for t = threadIdx.x : blockDim.x : N:
6:       exbuf[t] ← KB.prior[t]
7: sync threads
8: —          # Knowledge mapping
9: for each kl ∈ block.KLs:
10:      if IN[kl.input_idx] is empty:
11:           continue
12:      strip ← kl.strips[IN[kl.input_idx]]
13:      for t = threadIdx.x : blockDim.x : strip.len:
14:           e ← strip.entries[t]
15:           atomicAdd(exbuf[e.key], e.value+B)
16: sync threads
17: —          # Excitation reduction
18: obs ← exbuf[IN[block.input_idx]]
19: b ← threadIdx.x
20: for t = b+blockDim.x : blockDim.x : N:
21:      exbuf[b] ← max(exbuf[b], exbuf[t])
22: sync threads
23: ref ← threadReduceMax(exbuf[0:blockDim.x])
24: Score[blockIdx.x] = (ref - obs) / ref
```

3.5 Incremental Learning

Given the configured confabulation network, a KB was constructed for AnRAD from the input data streams. The learning process was to determine the weight of the knowledge links, i.e., the $p(s|t)$ values in Equation 1. This was achieved by collecting the statistics of the co-occurrences of the linked lexicon symbols. The probability was then calculated as $p(s|t) = cnt(s,t)/cnt(t)$, where $cnt(s,t)$ was the number of co-occurrences of the source s and the target t, and $cnt(s,t)$ was the occurrence of the target symbol t.

For probabilistic inference applications the above method worked fine because larger data samples generally gave more information of the most likely symbols. However, this was not true with anomaly detection, because the unlikely symbol was not necessarily made more distinguishable when the sample size increased. Zimek [14] determined that smaller datasets outperformed larger ones, and that constantly incrementing the co-activation counters can degrade the detection performance. Therefore, our framework used a mechanism named "episodic training", in which the co-activation counters were reset after time period T_{ep}. Then the excitation stored in the KB was updated by merging the new one into the previous episodes using the following equations:

$$ex^{E+1}(s,t) = \frac{ex^{E}(s,t) + ln[p(s|t)/p_0]}{E+1}, \tag{9}$$

$$ex^{0}(s,t) = \ln(\frac{p(s|t)}{p_0}), \tag{10}$$

where ex^E was the stored excitation at episode E. When equation (6) was substituted into equation (1), it resulted in:

$$el(t) = \sum_{k \in F_l} \{ \sum_{s \in S_k} [I(s) ex(s,t)] + B \}. \quad (11)$$

Before the first reset, i.e. ex^0, the result was the same as equation (1). This updating function resulted in an ensemble of temporal sub-samples.

3.6 Accelerating AnRAD

In terms of the computation recall complexity, AnRAD is similar to other conventional anomaly detection approaches. However, the AnRAD architecture, if parallelized, would significantly accelerate its processing speed. This section addresses the complexity of accelerating the AnRAD algorithm.

3.6.1 Complexity Analysis.

Accelerating the algorithm required an understanding of the processing bottleneck. The Confabulation (CFB) method was compared to some of the standard anomaly detection algorithms, including the incremental Local Outlier Factor (LOF), the Replicator Neural Network (RNN), and the Cross-Feature Analysis (CFA). Their computing time and the complexity to process a single Defense Advanced Research Projects Agency (DARPA) data stream are given in Table 1.

Table 1. Complexity Analysis

Model	Training time per frame	Training complexity	Recall time per frame	Recall complexity per frame
LOF	4854.9ms	$O(QN^2)$	684.1ms	$O(QNlogN)$
RNN	2916.9ms	$O(TNS)$	0.27ms	$O(S)$
CFA	4.30ms	$O(QHN)$	7.78ms	$O(QH)$
CFB	1.57ms	$O(NLF)$	243.1ms	$O(LDF)$

(Q – number of features; N – number of training samples; T – neural network iterations; S – neural network connections; H – decision tree height; L– key lexicon numbers; D – average key symbols; F – average knowledge link numbers)

The programs are all single threaded without intentional algorithmic optimizations. In terms of training, CFB is much faster than the others because at each frame it only updates a single entry of the affected knowledge link tables. Therefore it allowed real-time processing and incremental training without much optimization. However, the recall of confabulation was merely faster than the incremental LOF, whose complexity scaled with the volume of the training samples. Fortunately, the confabulation network has layered and massive parallel structure. And this can be exploited for performance acceleration with the help of today's multicore processors.

According to anomaly score equations (1) and (2), the complexity of detecting one instance is $O(LDF)$, where L is the number of key lexicons, D is the average number of symbols in one key lexicon, and F is the average number of knowledge links connected to one symbol. At node level, each key lexicon worked as an independent test, so L could be parallelized on multiple computing elements, e.g. Compute Unified Device Architecture (CUDA) blocks. At the

symbolic level, D and F induced the accumulation of link values from the knowledge tables to the candidate symbols. These operations were parallelized by either creating one CUDA thread for each knowledge link or by using vector processing.

In term of the storage space complexity, the confabulation model was dominated by $O(LFB)$, in which B was the average size of the knowledge link matrices. The actual memory requirement could be lower. For example, if lexicon R_1 and R_2 had connections to each other, then they could mirror the knowledge link. Also, features such as "shared links" could be adopted to reduce the KB size. The main optimization relied on B. Because most of the knowledge links were sparse matrices, a compact storage format was preferable.

3.6.2 Naïve Parallel Implementation.

The straightforward implementation design mapped each key lexicon to its own thread. Multiple threads were then run on a state-of-the-art multi-core CPU. This was feasible because the key lexicon computations were independent, and thus did not require significant synchronization. Also, there were usually a few dozen key lexicons, so the workload was sufficient to achieve high occupancy. As shown in Figure 13, a thread pool was allocated with a number of simultaneous threads. The system assigned key-lexicon computations to the available threads, or waited until all of the worker threads were occupied. The pool size was no less than $(2 * number\_cpu\_cores)$, so context switching was prevented. However, the limited number of available CPU cores prevented us from fully exploiting the structural parallelism of AnRAD.

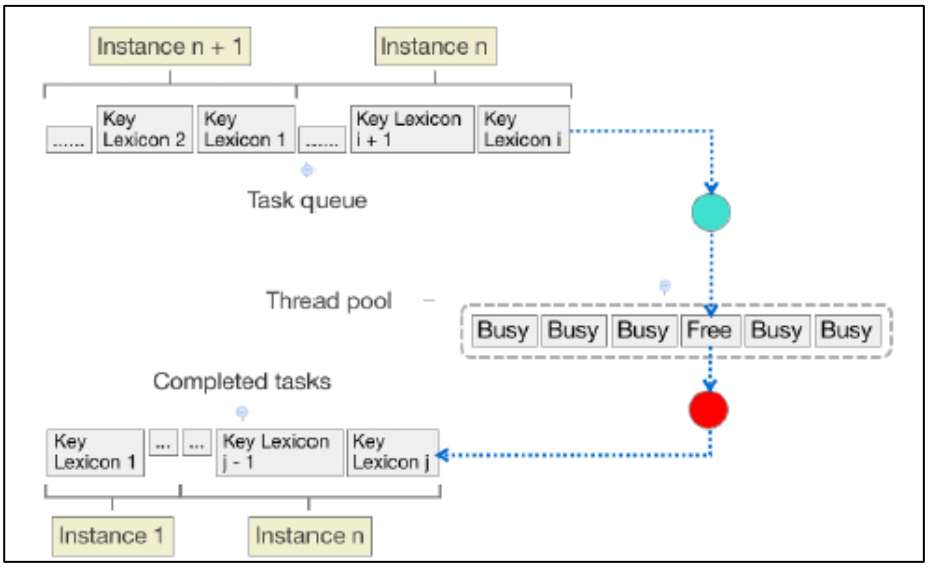

Figure 13. Thread Pool for CPU Multi-threading

GPUs also provided a potential option to fully parallelize the key-lexicon computations. Most GPUs may have more cores than most state-of-the-art CPUs. A simple design directly moved the aforementioned CPU threads to GPU cores using kernel Algorithm 4. In this implementation, each CUDA thread handled one key lexicon, and distinct CUDA blocks processed concurrent input streams.

Algorithm 4 Naive Recall Kernel

```
1: procedure naive_recall(KB, IN):          # KB: knowledge base; IN: input symbols
2: ref ← 0
3: for symbol t ∈ key lexicon KB.lexicons[threadIdx.x]:
4:     initialize el
5:     for kl ∈ KB.lexicons[threadIdx.x].links:
6:         el += kl[t][IN[kl.input_idx]] + B
7:     ref = max(ref, el)
8:     obs = el if t == IN[threadIdx.x]
9: Score[threadIdx.x] = (ref - obs) / ref
```

This naïve acceleration had two major problems. First of all, it had inefficient KB management. In the CPU implementation the knowledge link matrices were stored in hash tables to provide near constant lookup time. However, on the GPUs concurrent random accesses affected the memory bandwidth and induced stalls. Secondly, it caused imbalanced workloads among threads. The number of symbols in key lexicons was determined by the nature of the targeted application and varied significantly. Thus, different key lexicons may sometimes introduce different workloads. Such workload imbalance produced serious control divergence, since the CUDA threads were executed in warps. If the threads had to wait for their neighbors' outstanding workloads, the overall acceleration was diminished.

These limitations of the naïve implementation motivated us to modify three aspects of the AnRAD system to provide GPU acceleration: KB management, workload balancing and finer-grained parallelization.

3.6.3 In-memory Knowledge Bases.

The KB of the confabulation network was flattened and stored in the device memory. There could be multiple KBs on the same device, each associated with a knowledge link. Figure 14 shows the memory layout of one KB. The KB maintained a "Block List", each entry of which corresponded to a key lexicon. Based on the size of its symbol list and the amount of available shared memory, a key lexicon could be divided into multiple blocks.

In addition to number of symbols and number of source lexicons, a block entry also stored the location of a "KL List" which described the incoming knowledge links of the key lexicon. A KL entry pointed to the list of symbols in the source lexicon for that knowledge link, and provided the starting address of the knowledge link matrix. Note that the knowledge link matrix was very sparse, for memory reduction the matrix was stored in a *list of lists* (*LIL*) format.

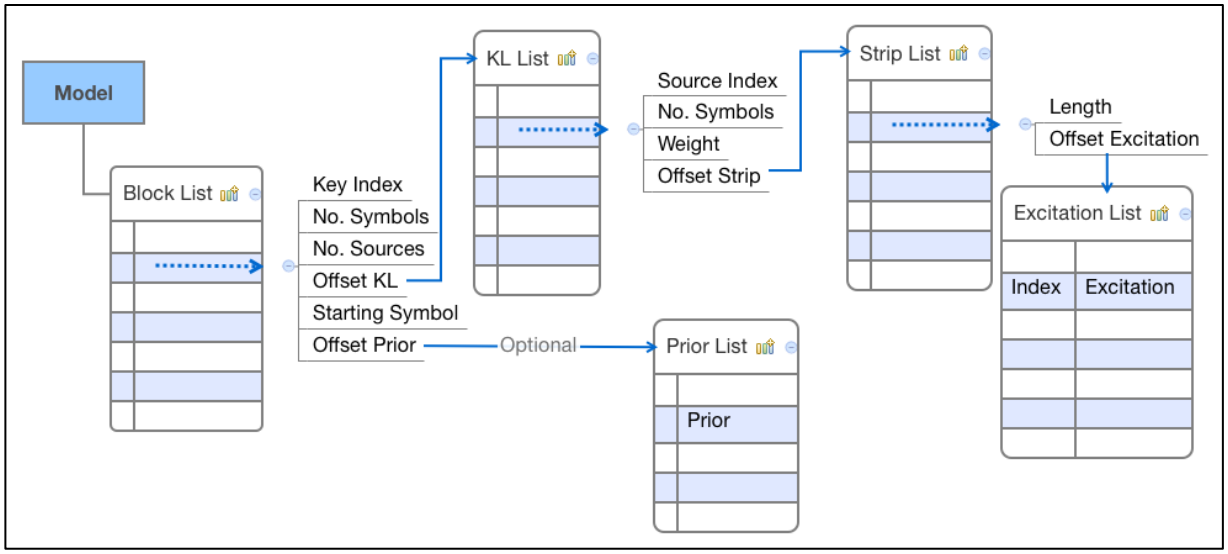

Figure 14. In-memory Knowledge Base Layout

Each LIL had a strip list, which corresponded to rows in the matrix. Each entry in the strip list pointed to an excitation list, which corresponded to non-zero elements in the row. The matrix was arranged such that each row corresponded to a symbol in the support lexicon, and the column represented a symbol in the key lexicon. The input of support lexicons were invariant when the detect process calculated the excitation level of all symbols of the key lexicon for a given scene. Hence only one excitation list needed to be loaded for each knowledge link. The column and row arrangement of knowledge matrix insured that the algorithm accessed knowledge values in continuous address. Finally, each block entry also contained addresses to the prior probabilities of the key symbols that were needed for the calculation of Equation (8).

The size of the trained KBs for a single detection zone is plotted in Figure 15. The naive implementations stored raw knowledge matrices and quickly scaled the memory usages as the training data increased in size. On the other hand, the optimized memory layouts compressed the sparse knowledge links and significantly reduced the memory consumptions. The shared link feature included knowledge links that connected the different neighbor vehicles that shared the same probability matrices. This not only made the nodes of interactive features more general, but also reduced the memory usage. For this example, compared to naive and LIL-only (Optimized) storages, enabling shared memory (Shared) reduced the knowledge base size to less than 20MB.

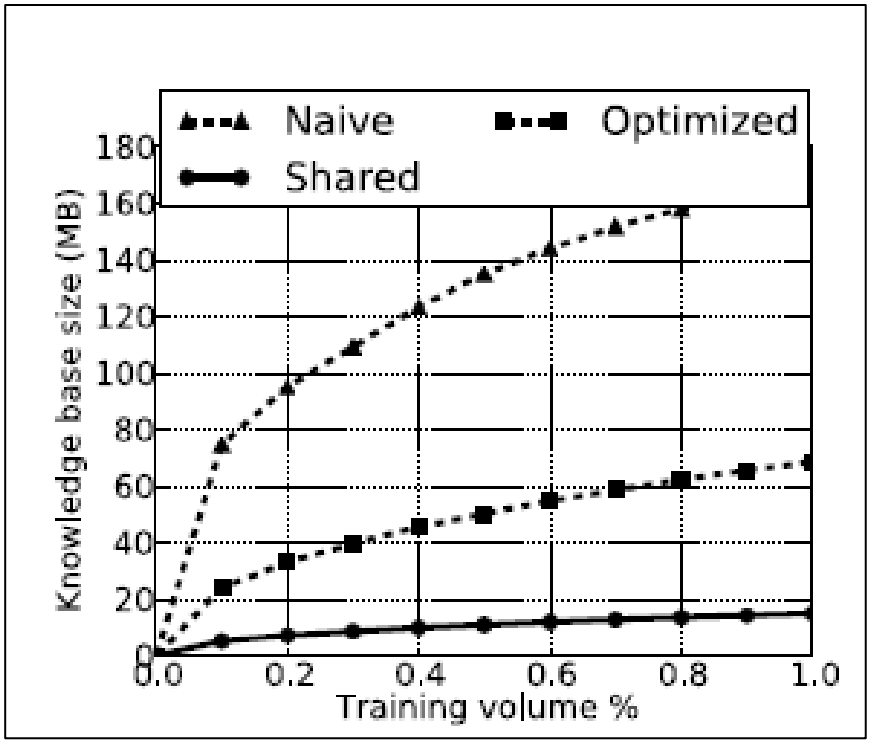

Figure 15. Memory Usage of Individual Models

3.6.4 Workload Balancing.

Instead of mapping the key lexicons to individual threads, a CUDA block was assigned to each key lexicon. Each CUDA block consisted of multiple threads. The blocks were dynamically schedulable, and uneven workloads among different lexicons did not introduce control divergence. Also, multiple threads in the CUDA block contributed to the anomaly score of the same key lexicon. This exploited the layered parallel structure, i.e. optimized the D and F factors.

During the system initialization, the trained KBs were flattened and loaded to the GPU. The input streams were then organized into the corresponding format and then were dynamically sent to the devices at each frame. One CUDA block either computed the anomaly score of one key lexicon or a part of a larger lexicon with many symbols.

3.6.5 Extension to Xeon Phi Co-processor.

The memory layout and computation process was also applied to a Xeon Phi co-processor. The in-memory KB was also used when the co-processor was in the off-load mode. However, the way that the workload was mapped needed to be changed. Typically a Xeon Phi KNC chip has fewer physical cores than the NVIDIA GPGPU, but each of the Xeon Phi cores is a fully featured processor, and thus more powerful than the GPU shadows.

In particular, each Xeon Phi core was equipped with a 256-bit vector engine, which can perform the mapping reduction process. Therefore, the lexicon-wised CUDA block computations were mapped to individual Open Multi-Processing (OpenMP) threads. Within each OpenMP thread the vector unit replaced the function of the original CUDA threads.

4 RESULTS AND DISCUSSIONS

Experiments were carried out to evaluate the basic AnRAD system, the effectiveness of the self-structuring algorithm, and the speed-ups of the optimized implementation. In the following set of experiments the AnRAD system detection accuracy was determined. To evaluate the effectiveness of the framework, in addition to the GMTI dataset [15] for large area surveillance, two other publicly available anomaly detection datasets were analyzed. The second dataset was composed of extracted package streams from the DARPA 1998 Intrusion Detection Program [16]. The third dataset was the ADFA-LD contained system call sequences [17-19] of benign and malicious programs.

4.1 AnRAD for Traffic Monitoring of Manually Structured Data

The detection performance of the AnRAD system was evaluated with manually constructed confabulation network data that was described in Section 3. A zone of 500x500 meter^2 with moderate traffic density was randomly selected from the total monitored area. The training data was for 240 minutes of normal traffic. The testing data included 10 minutes of normal traffic data. Abnormal events representing typical hazardous vehicle activities had been manually inserted into the data. The abnormal events included cars deviating from the road, speeding, tailgating, 18-wheeler trucks running at abnormal speeds, and cars that unexpectedly "Start/Stop" in the middle of the road.

Figure 16-19 are the anomaly scores of selected key lexicons of all of the target vehicles in the testing area during the time when abnormal events occurred. This anomaly value is a representation of how anomalous the data set was. In historical modes, the value of the score isn't as important as the relative height of the peak of the score line. The X-axis in each of these figures gives the indices of vehicles. The Y-axes are the magnitudes of the anomaly scores. Each figure corresponded to a type of abnormal activities.

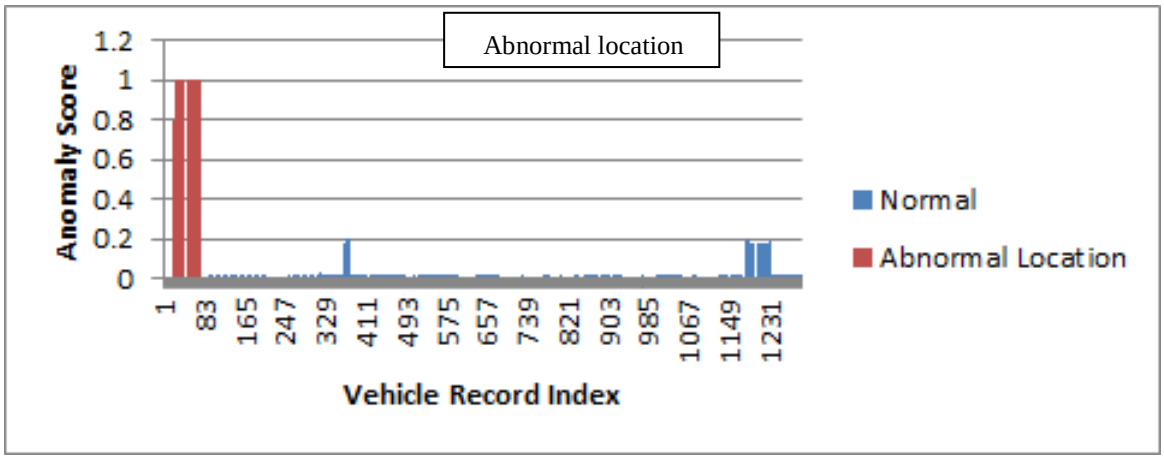

Figure 16. Anomaly Score of a Location Key Lexicon

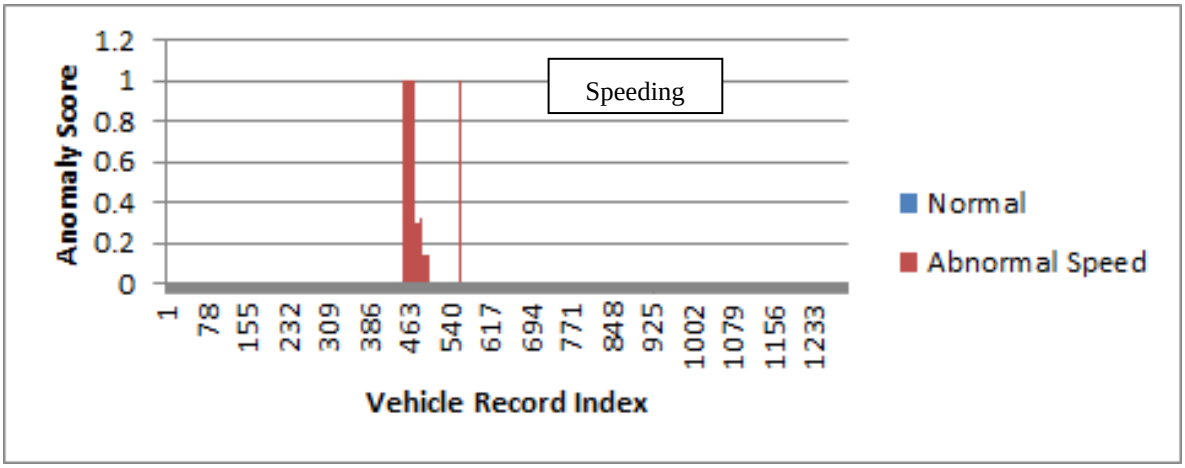

Figure 17. Anomaly Scores of Speed Key Lexicons

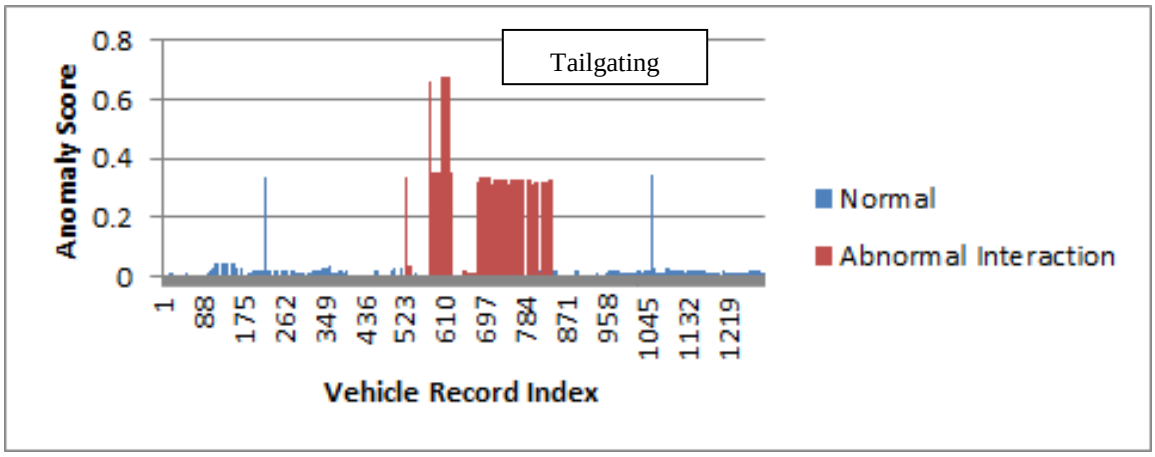

Figure 18. Anomaly Scores of First Neighbor Pair Lexicons

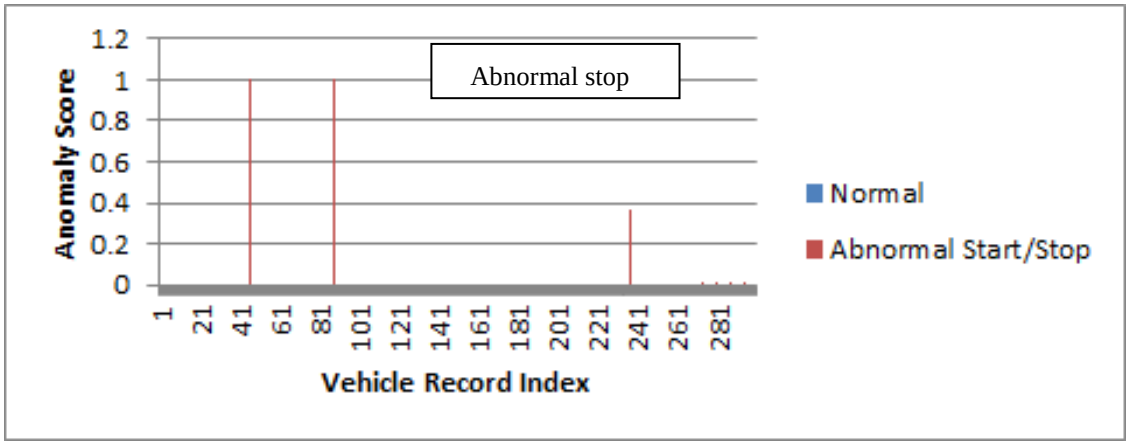

Figure 19. Anomaly Scores of the Speed Lexicon for "Start/Stop" Events

The anomaly scores of the manually-inserted abnormal targets are highlighted in red in each figure. The anomaly scores of the normal vehicles are in blue. The anomaly scores in red were significantly higher than the blue ones, and could be readily detected by a decision threshold. Furthermore, the anomaly scores demonstrated an obvious temporal continuity for most of the abnormal events categories. The exception was the abnormal "Start/Stop" of vehicles, which gave short spikes only when the moving status changed. These results also demonstrated that different key lexicons corresponded to different types of anomalies. For example, a high anomaly score in the neighbor pair lexicon indicated a tailgating event or some other unusual relation between two vehicles.

The anomaly scores for a set of anomaly data were compared using the AnRAD framework and a traditional Bayesian model. For this data an 18-wheeler by itself was quite common in the zone, but this type of vehicle becomes abnormal when it was driven at a speed that was normally observed for sedans. From the anomaly scores as shown in Figure 20, the AnRAD framework was more effective in detecting these anomalies. The AnRAD anomaly scores for such events were 20% higher than those calculated with Bayesian model, which resulted in the AnRAD model providing a higher detection probability than the Bayesian model.

4.2 AnRAD Accuracy

4.2.1 Abnormal Vehicle Behavior Detection.

For this determination vehicle traces were obtained from an area road network. Each record contained the vehicle location, speed, type and a timestamp. Our preprocessor extracted the interactive features including distances and velocity angles between vehicles and their neighbors. The original features and the interactive measures together formed 10 primary features. The self-structuring procedure selected 44 key lexicons out of the 2548 possibilities. The maximum order of feature-wise pooling was set to 5, and the maximum order of temporal pooling was 3. The traffic records were generated at one-second sampling intervals in four randomly picked zones. The training stage consumed 240 minutes of traces, and another 10-minute trace was used as the test set. Among the test data there were 179 vehicles that, without intentional modification, were used as the negative cases, and 22 manually created anomalies of different categories that were the positive cases.

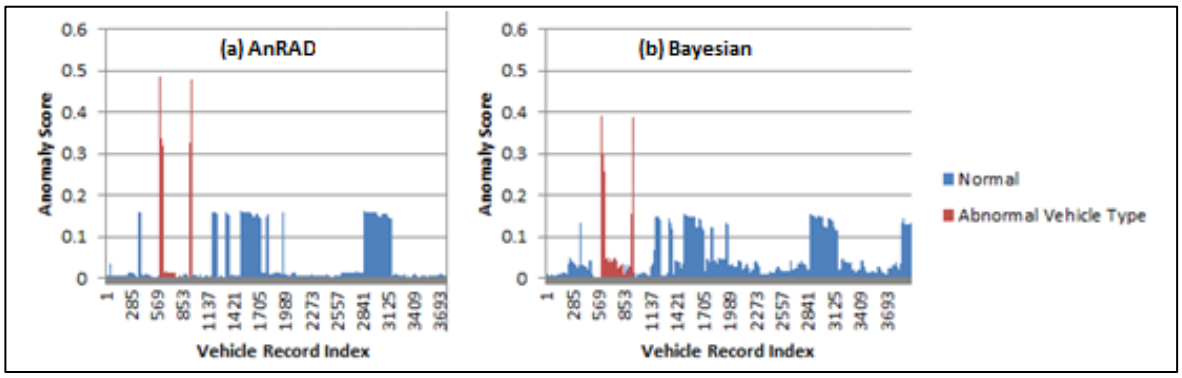

Figure 20. Anomaly Scores for (a) AnRAD and (b) Bayesian

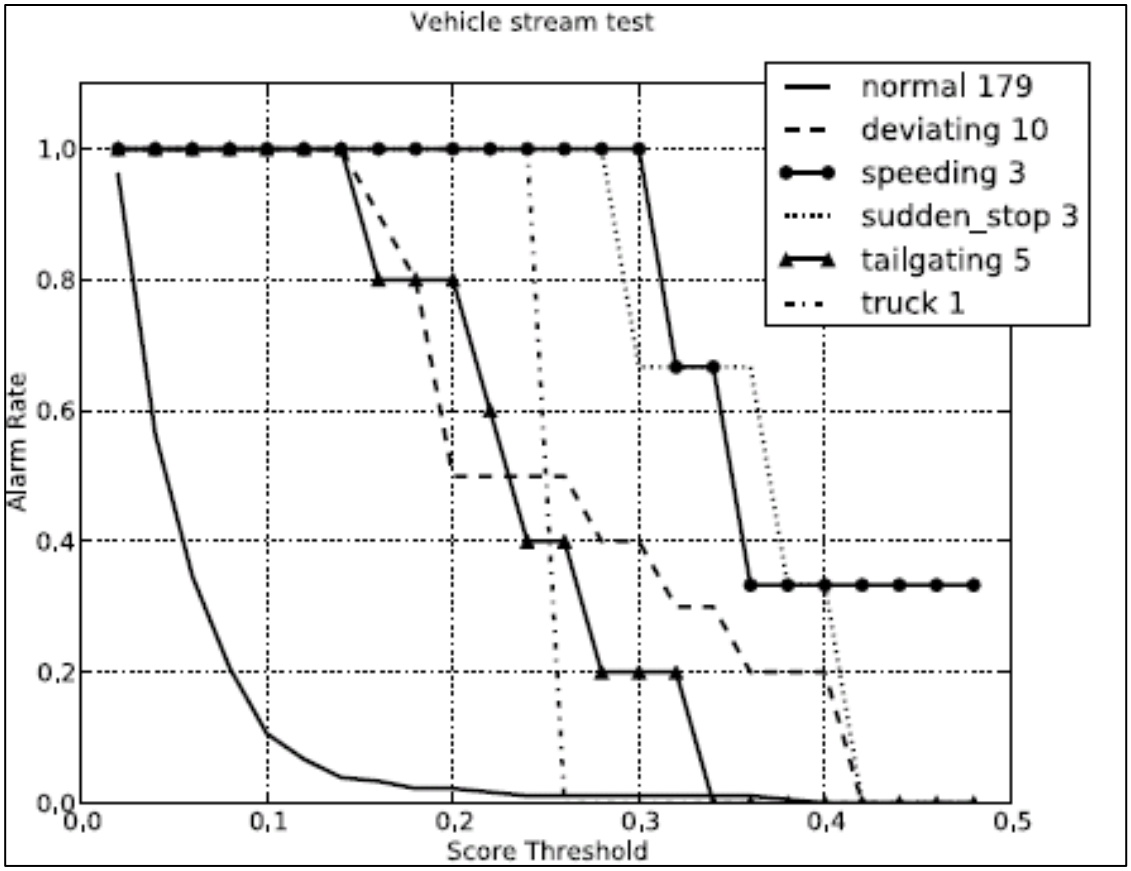

Figure 21. Alarm Rate v. Score Threshold of Vehicle Monitoring

In Figure 21 is shown the anomaly class detection results. The Y-axis is the alarm rate and the X-axis is the network anomaly score threshold. It was observed that the normal vehicles generated a much lower alarm rate compared to abnormal ones. When the threshold was 0.14, a 100% anomaly detection rate was achieved when the false positive rate was 10e-2. Therefore, for vehicle anomaly detection tasks, the framework leaves a wide margin to trade between detection and false alarms. The self-structured network was able to detect abnormal behaviors such as tailgating, deviating from driveway and speeding.

Table 2 Correlation between Anomalies and Outstanding Nodes

Anomaly	Top 3 Outstanding Nodes
sudden stop	1. ⟨speed⟩; 2. ⟨neighbor(1).speed⟩; 3. ⟨neighbor(1).distance⟩
speeding sedan	1. ⟨speed⟩; 2. ⟨neighbor(1).speed⟩; 3. ⟨neighbor(1).distance⟩
tailgating	1. ⟨longitude, neighbor(1).distance⟩; 2. ⟨longitude, latitude, speed, direction⟩; 3. ⟨longitude, speed, direction⟩
deviating from driveway	1. ⟨longitude⟩; 2. ⟨longitude, latitude, neighbor(1).direction⟩; 3. ⟨longitude, speed, direction⟩
speeding truck	1. ⟨vehicle_size, longitude, longitude^{-2}⟩; 2. ⟨latitude, speed, neighbor(1).distance, neighbor(1).speed⟩; 3. ⟨vehicle_size, longitude, longitude^{-1}⟩

The AnRAD framework provided the reasoning ability in that the anomaly decisions were explainable by the introspection of the anomaly scores of the key lexicons. For instance, Table 2 showed the relationship between the key lexicons and the anomaly classes. In this example, key lexicons that generated an anomaly score of higher than 0.8 were defined as "outstanding". The different anomaly outstanding occurrences were counted, and three largest lexicons were noted. For speeding and sudden stops, i.e., these anomalies were closely correlated. Our analysis also showed that the most outstanding lexicon for this type of anomaly was <*speed*>.

Tailgating happens when one vehicle quickly approaches another vehicle. This was detected when the composite lexicons of speed and distance to the first neighbor had an increased anomaly score. Anomalies such as deviating from the road were revealed by high anomaly scores in the coordinates-related lexicons. A truck was determined to be speeding even when its speed was normal for a sedan. Such behavior was flagged by the composite lexicon of vehicle size and its displacement in consecutive frames. These examples showed that the AnRAD framework used features to determine additional classes of anomalies without additional training labels or domain knowledge.

4.2.2 Network Data Intrusion Applications.

The AnRAD framework was also tested with fully labeled datasets and compared with other baseline methods. The baseline algorithms considered were: incremental Local Outlier Factor (LOF) [20], which is a density-based method; Replicator Neural Network (RNN) [21], which is a classification-based method; and Cross-Feature Analysis (CFA) [22] with Classification and Regression Tree (CART) decision trees (rule-based method). The baselines did not have all of the AnRAD functionality such as reasoning and incremental training, here only the detection performances were considered.

The first dataset was processed from the 1998 DARPA Intrusion Detection Evaluation Data Set. For each Internet Protocol (IP) address pairs, traffic statistics were recorded per 300 msec. per frame. In total 21 primary features were extracted from the raw files. Some examples of the features were bytes from client to server (or server to client), service ports, and the number of clients connected with a server.

We did not use the session-oriented Knowledge Discovery and Data Mining (KDD) 99 dataset because we investigated the concurrent data streams rather than the session-oriented data points, and our processing also leveraged less attack specific domain knowledge. The self-structuring network picked 123 key lexicons out of the 446,320 possibilities given the max order equals five for both the feature-wise and the temporal pooling. For training, normal streams from the seven weeks of training data were randomly sampled, and 20,000 frames were selected. The negative class for this test had another 7000 streams, and all the attacks (422 streams, 24 categories) in the seven weeks formed the positive class. The moving window size was five frames for all of the methods.

The Receiver Operation Curves (ROCs) for the DARPA dataset are plotted in Figure 22, with the X-axis representing the false alarm rate and the Y-axis representing the true detection rate. Note that the true positive rates were averaged across the anomaly categories to prevent the results from being biased by the larger classes. The AnRAD method outperformed the network that had an equal number of randomly selected zones (Random). The AnRAD method also obtained the best "Area Under the Curve" (AUC), which is a measure of accuracy, as compared to the Incremental Local Outlier Factor, the Replicator Neural Network and the Cross Feature Analysis decision tree method. The results demonstrated that the AnRAD method had the advantage in the tradeoff between false alarms and detection rates. This was because the AnRAD method was able to capture implicit patterns, while the general baseline methods did not. In this example the LOF and the RNN methods outperformed the CFA decision tree approach because they worked better with continuous features.

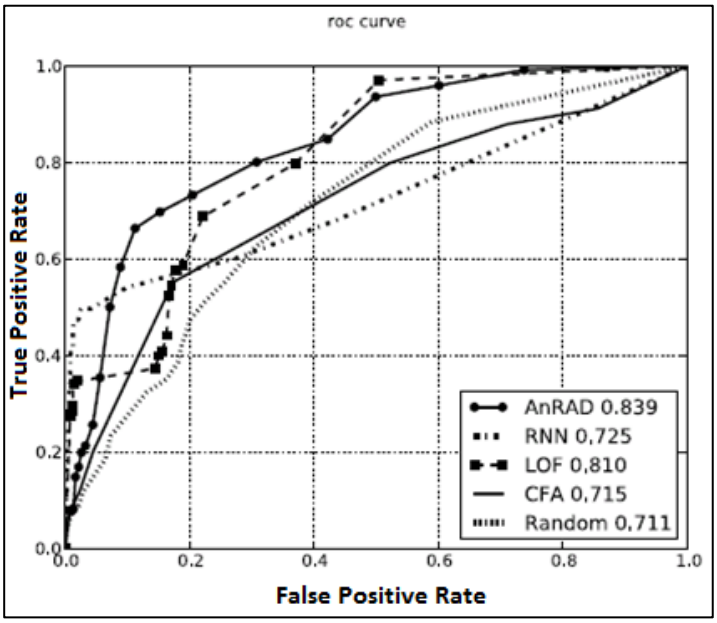

Figure 22. DARPA Dataset ROC Analysis

The second dataset was system call sequences from the ADFA-LD dataset, which included discrete features that were generated from a Linux local server configured to represent a contemporary computer system. The training data consisted of less than 20,000 system calls. The testing data had 6000 sequences from the validation set and 746 sequences from the attack set. To enable LOF the Levenshtein Edit Distance[4] was used; for the neural network method 100 frequent and orthogonal system calls were sampled from the training set as the template points, and the input layer of the network received the call distances to these templates. The moving window size was set to six consecutive calls. Because this dataset had fewer primary features, we also evaluated the confabulation network with all of the 41 possible nodes and full connections (Full).

From the ROC analysis curves is shown in Figure 23 the AnRAD and the decision tree methods outperform the other two. This was because the latter two approaches did not adapt as well to the purely categorical features. The decision tree had a marginally better AUC score, but it suffered from over fitting: its performance was reasonable at the high-detection-rate regions, but its false positive rates were 19% and greater. Also, the self-structured 10-node network was determined to have a better performance than the Full configuration.

The comparisons in this section demonstrated that the AnRAD framework's detection performances were equal to or superior to the classical methods. The AnRAD method is the only method that also provided incremental training, transparency and adaption to both continuous and categorical data.

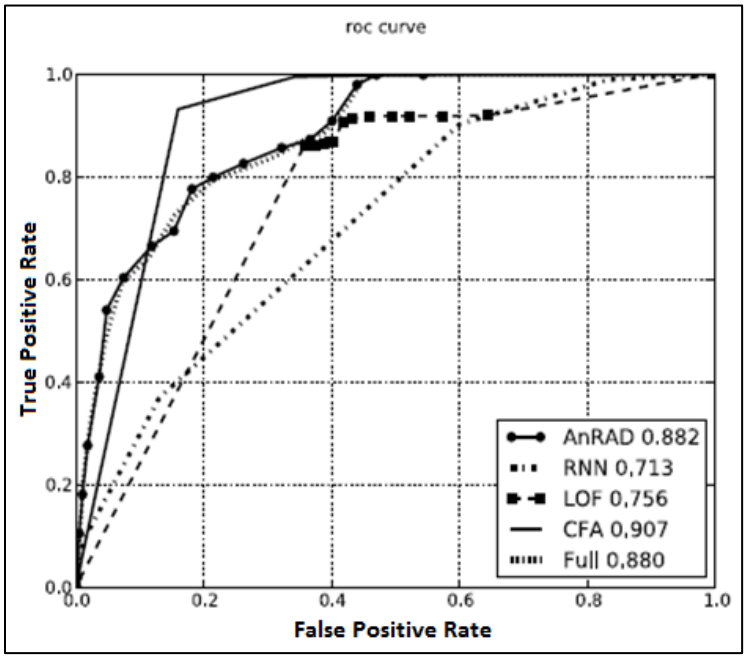

Figure 23. ADFA-LD Dataset ROC Analysis

[4] The minimum number of single-character edits required to change one word into another.

4.3 Incremental Learning Benefits

In the training sequences experiments data was streamed into the system in an incremental fashion to verify that the AnRAD model evolved and improved as additional training data become available. In this test a detection zone was selected, and the training data was gradually presented to the system in 10-minute-long segments. After every incremental training step a 10-minute-long normal traffic sequence that was different from any of the training segments was used for the testing, and the numbers of detected anomalies in any key lexicons were reported. Since the testing sequence consisted of normal vehicles, these anomaly counts were false alarms, and were expected to decrease with additional training.

In Figure 24 is shown that the false alarm rate was reduced with increased training. The false alarm rate was calculated as the ratio of the reported anomalies over the total number of checked instances. Each line represented a false alarm rate of one category of anomaly category (i.e., a key lexicon). With insufficient training, e.g. 10 minutes of the training sequence, the false alarm rate was as high as 60%. With additional training, the false alarm rates quickly decreased. The system reported near zero false alarms after 150 minutes of data training. This indicated that the model has been incrementally updated, and that it had become more accurate with additional training.

Another benefit of the incremental training was that it diluted the impact of the false training data. Training used clean samples, i.e. anomalies were not intentionally inserted. However, in the real unsupervised case, there was no guarantee of the training set quality. So it was important that the framework incrementally improved the KB quality.

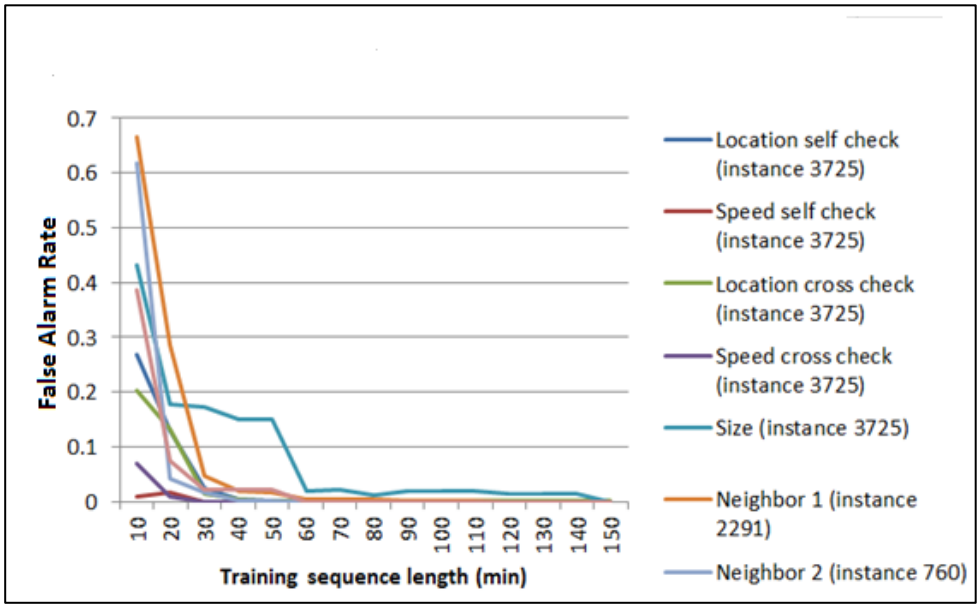

Figure 24. False Alarm Rate v. Training Time

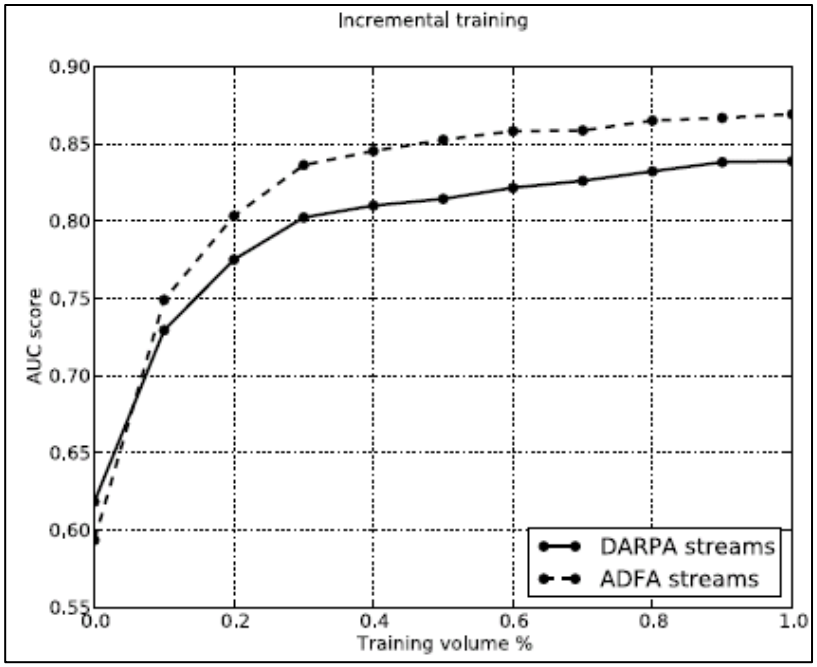

Figure 25. AUC Scores v. Training Sizes

In the next experiment the train confabulation networks were first trained with anomalous data. Then the clean training sets were segmented into 10 episodes and added into the KB one by one. At each stage the model tested the same evaluation set, and the AUC scores were collected. In Figure 25 for both the DARPA and the ADFA datasets the AUC scores increased as the incremental training increased. As more data streams were received, the AnRAD framework was able to update and further correct for earlier erroneous knowledge. The detection performance continuously improved with new and better training data.

4.4 Performance Evaluations

To test the effectiveness of the fine-grained parallelization, datasets were implemented on a CPU with single threading and multi-threading, a naive GPU and an optimized GPU, and a Xeon Phi KNC. The four designs were compared using single testing streams from the normal vehicle classes of the DARPA and ADFA datasets. For CPU multi-threading programs were run on an Intel Xeon W5580 with 16 cores at 3.20 GHz frequency. For the GPU implementations the device used was a NVIDIA Tesla C2075 with 448 CUDA cores at 1.15 GHz and 6 GB device memory. U_{max} = 1536 and *blockDim.x* = 192 were selected to achieve full occupancy. The Intel co-processor implementation was on a Xeon Phi 5100 with 60 cores, 1.053 GHz processor speeds, and a memory capacity of 16 GB. A maximum of 240 threads were allocated.

Table 3. Runtime Comparison Results

Implementation		DARPA	ADFA-LD	Vehicle
CPU 1-thread	time	200.5ms	10.8ms	78.6ms
	speedup	1X	1X	1X
CPU 16-thread	time	25.7ms	3.45ms	12.2ms
	speedup	7.8X	3.1X	6.4X
GPU naive	time	146.9ms	17.7ms	150.1ms
	speedup	1.4X	0.61X	0.52X
GPU optimized	time	0.210ms	0.0232ms	0.178ms
	speedup	**955X**	**466X**	**442X**
Xeon Phi KNC	time	4.04ms	0.242ms	3.27ms
	speedup	49.6X	44.6X	24.0X

The runtime comparison results in Table 3 showed that the CPU implementation with 16 threads resulted in a 3-8 times speedup compared to the serial baseline implementation. It did not linearly scale with the thread number due to the memory stalls caused by the concurrent memory accesses. The GPU naive implementation provided marginal improvements, or ran slower than the single-thread baseline, because imbalanced workloads across the threads produced control divergence. The optimized GPU implementation provided 442-955 speedups over the baseline methods. The optimized kernel fully exploited the concurrent structure of the confabulation model and avoided control divergence. Also, the memory access pattern improved the cache performance. The Xeon Phi implementation had measured speedup improvement of 24-50 times. The reason that Xeon Phi implementation was not as fast as GPU is that the single testing streams did generate enough workload for the 240 threads. The GPU offered better responsiveness for small and randomly arrived service requests, while the processing power of Xeon Phi will be sufficiently utilized by large workload batches.

5 CONCLUSIONS

In this project a high performance computing-based neuromorphic anomaly detection framework was developed that provided real-time processing for concurrent data streams. The autonomous anomaly recognition and detection (AnRAD) framework was based on cogent confabulation, which is a probabilistic inference model that mimicked human information processing. It extracted the conditional probability among symbolic representations of features in an unsupervised environment.

Large areas of surveillance data were first partitioned into smaller zones that were then independently processed. A Knowledge Base (KB) was built for each zone by adding traffic records into properly modeled knowledge networks. When new traffic information was received, anomaly scores were calculated by means of the likelihood-ratio test for the observed events. Events with high anomaly scores were then marked as potential anomalies, and alarms were sent to a human observer. The unique features of this platform are summarized as follows:

1. The model was able to handle a large volume of vehicle traces over a large area. Large areas had not been considered in previous works. The surveillance area was partitioned-based on the traffic density information extracted from the training data. In this way, the computation load was balanced, and the inference model was more accurately constructed.

2. The confabulation-based model had a lower complexity for both the training and the recall. The system was trainable while operational, and this enabled continuous improvements to the KB quality.

3. By proper modeling the system was capable of capturing contextual information among vehicles and their neighbors. Abnormal events such as tailgating that were caused by interactions among vehicles were detected. Such events were not considered in previous research.

4. The overall system had a hierarchical architecture, and the workloads in each level of the hierarchy were inherently parallel. A GPU-based parallel implementation was adopted that achieved computation acceleration of the AnRAD system.

The anomaly detection confabulation model is application specific. The neuron nodes (lexicons) and the synapses (knowledge links) between them were reconfigurable for different applications. The AnRAD system enabled the automatic construction of a confabulation network. It concretely learned from the data a succinct set of nodes that represented original features or combinations of features. Each node was associated to a lexicon, which recorded the symbolic representations of the possible inputs. The links between nodes were also learned from the initial data. Given the learned network configuration, further incoming data streams were used to incrementally refine the weight of the knowledge links, which was the conditional probability between the lexicon symbols.

With the self-structuring technique, the AnRAD framework was generalized to a wide range of applications. In addition to road-traffic monitoring, it was also applied to network intruder detection and program control flow monitoring.

6 REFERENCES

[1] Hecht-Nielsen, R., **Confabulation Theory: The Mechanism of Thought**, Springer-Verlag, Berlin, Germany, 2007.

[2] Chandola, V., Banerjee, A., and Kumar, V., "Anomaly Detection: A Survey," *ACM Computing Surveys* (CSUR), 41(3), Article 15, July 2009.

[3] Roth, V., "Outlier Detection with One-class Kernel Fisher Discriminants," *Proc. of the Conference on Advances in Neural Information Processing Systems 17*, 2004.

[4] Hawkins, S., He, H., Williams, G.J. , and Baxter, R.A. , "Outlier Detection using Replicator Neural Networks," *Proc. of 5th International Conference on Data Warehousing and Knowledge Discovery*, 2002.

[5] Yu, J.X., Qian, W., Lu, H., and Zhou, A., "Finding Centric Local Outliers in Categorical/Numerical Spaces," *Knowledge and Information Systems*, Vol. 9(3), pp. 309-338, March, 2006.

[6] Huang, H., Mehrotra, K., and Mohan, C.K., "Rank-based Outlier Detection," *Journal of Statistical Computation and Simulation*, 83(3), pp. 518-531, 2013.

[7] Sebyala, A.A., Olukemi, T., and Sacks, L., "Active Platform Security through Intrusion Detection using Naive Bayesian Network for Anomaly Detection," *Proc. of the 2002 London Communications Symposium*, 2002.

[8] Fu, Z., Hu, W., and Tan, T., "Similarity based Vehicle Trajectory Clustering and Anomaly Detection," *Proc. of IEEE International Conference on Image Processing*, 2, pp. 602-605, 2005.

[9] Sheng, H., Li, C., Wei, Q., and Xiong, Z., "Real-time Detection of Abnormal Vehicle Events with Multi-feature over Highway Surveillance Video," *Proc. of 11th International IEEE Conference on Intelligent Transportation Systems*, pp. 550-556, 2008.

[10] Bhardwaj, A., Farooq, F., Cao, H., and Govindaraju, V., "Topic Based Language Models for OCR Correction," *Proceedings of the Second Workshop on Analytics for Noisy Unstructured Text Data*, pp. 107-112, 2008.

[11] Qiu, Q., Wu, Q., Bishop, M., Pino, R., and Linderman, R.W., "A Parallel Neuromorphic Text Recognition System and Its Implementation on a Heterogeneous High Performance Computing Cluster," *IEEE Transactions on Computers*, 62(5), pp. 886-899, 2013.

[12] Mitra, P., Murthy, C. and Pal, S.K., "Unsupervised Feature Selection using Feature Similarity," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3), pp. 301–312, 2002.

[13] Tanenbaum, A.S., **Computer Networks**, 4th Edition, Prentice Hall, New Jersey, 2002.

[14] Zimek, A., Campello, R.J., and Sander, J., "Ensembles for Unsupervised Outlier Detection: Challenges and Research Questions a Position Paper," *ACM SIGKDD Explorations Newsletter*, 15(1), pp. 11–22, 2014.

[15] O'Neil, S.D., "Estimating Road Network using Archived GMTA Data," *IEEE Proceedings Aerospace Conference*, Vol. 4, pp.1865-1871, 2001.

[16] Lippmann, R.P., Fried, D.J., Graf, I., Haines, J.W., Kendall, K.R., McClung, D., Weber, D., Webster, S.E., Wyschogrod, D., Cunningham, R.K., *et al.* "Evaluating Intrusion Detection Systems: The 1998 DARPA Offline Intrusion detection evaluation," *In DARPA Information Survivability Conference and Exposition*, 2000. DISCEX'00. Proceedings, 2, pp. 12–26. IEEE, 2000.

[17] Creech, G. and Hu, J., "Generation of a New IDS Test Dataset: Time to Retire the KDD Collection," *IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 4487–4492, 2013.

[18] Creech, G. and Hu, J., "A Semantic Approach to Host-based Intrusion Detection Systems using Contiguous and Discontiguous System Call Patterns," *IEEE Transactions on Computer*, 63(4), pp. 807–819, 2014.

[19] Xie, M., Hu, J., Yu, X. and Chang, E., "Evaluating Host-Based Anomaly Detection Systems: Application of the Frequency-Based Algorithms to ADFA-LD," *Network and System Security*, 8792, pp. 542-549, 2014.

[20] Pokrajac, D., Lazarevic, A., and Latecki, L.J., "Incremental Local Outlier Detection for Data Streams," Computational Intelligence and Data Mining," *IEEE Symposium on Computational Intelligence and Data Mining*, pp. 504–515, 2007.

[21] Hawkins, S., He, H., Williams, G. and Baxter, R., "Outlier Detection using Replicator Neural Networks," 5th International Conference on *Data Warehousing and Knowledge Discovery*, pp. 170–180. Springer, 2002.

[22] Cabrera, J.B., Gutie´rrez, C., and Mehra, R.K., "Ensemble Methods for Anomaly Detection and Distributed Intrusion Detection in Mobile Ad-Hoc Networks," *Information Fusion*, 9(1), pp. 96–119, 2008.

APPENDIX

The output of the Ground Moving Target Indicator (GMTI) was in the Earth-Centered, Earth-Fixed (ECEF) representation. To facilitate the display and analysis, the input data was converted from the ECEF format to a Geodetic format using the WGS84 earth model. The conversion algorithm is shown in Figure 26.

AmosCoord_ECF2Geo
Input arguments:
double *x, y, z*; //*ECF coordinates in meters;*
Outputs:
double *lat, lng, alt*; //*Latitude and longitude in degrees, altitude in meters*
begin
while (lat has not converged)
begin
$lng = tan^{-1}(\frac{y}{x})$; // *Calculate longitude*
$lat' = tan^{-1}(\frac{z}{(1-f)p})$; // *Guess of latitude and reduced latitude (lat'),*
$lat = tan^{-1}(\frac{z+\frac{e^2(1-f)}{1-e^2}a\sin^3(lat')}{p-e^2acos^3(lat')})$; //*in which* $p = \sqrt{x^2+y^2}$, f = *0.0033528106718309896 //(flattening), e2 = 0.006694380004260827 (first //eccentricity squared) and a = 6378137.0 (equatorial //radius), in accordance to WGS84 earth model.*
$lat' = tan^{-1}(\frac{(1-f)\,sin(lat)}{cos(lat)})$; // *After the guess, lat' is re-calculated*
end
$alt = pcos(lat) + \left(z + e^2Nsin(lat)\right)\sin(lat) - N$; // *Where* $N = a/\sqrt{1-e^2sin^2(lat)}$.
end

Figure 26. Converting ECEF Data to Geodetic Format

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

ADFA-LD	Australian Defence Force Academy - Linux Dataset
AUC	Area Under Curve
AnRAD	Anomaly Recognition and Detection
CART	Classification And Regression Tree
ΔL	Center Displacement
CFA	Cross-Feature Analysis
CFB	Confabulation
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DARPA	Defense Advanced Research Projects Agency
ECEF	Earth-Centered, Earth-Fixed
GMTI	Ground Moving Target Indicator
GPU	Graphics Processor Unit
IP	Internet Protocol
KB	Knowledge Base
KDD	Knowledge Discovery and Data Mining
KL	Knowledge Links
L	Target Location
ΔL	Center Displacement
Lexicon	Features that were used to describe behaviors
LOF	Local Outlier Factor
OpenMP	Open Multi-Processing
RNN	Replicator Neural Network

S	Target Size
SAR	Synthetic Aperture Radar
SVM	Support Vector Machine
V	Target Velocity
ΔV	Center Displacement